UNIVERSITÉ DE MONTRÉAL

# ADAPTIVE COMPUTING SYSTEMS FOR AEROSPACE

JACOPO PANERATI
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE INFORMATIQUE)
MAI 2017

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

ADAPTIVE COMPUTING SYSTEMS FOR AEROSPACE

présentée par : PANERATI Jacopo
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

Mme NICOLESCU Gabriela, Doctorat, présidente
M. BELTRAME Giovanni, Ph.D., membre et directeur de recherche
M. PAL Christopher J., Ph.D., membre
M. FIERRO Rafael, Ph.D., membre externe

# DEDICATION

*To caffeine*
*and all the medications*
*that made this possible.*

## ACKNOWLEDGMENTS

This work was only made possible by the continued support of the many people whose paths I crossed over the course of recent years. I want to thank Giovanni Beltrame, Alain Fourmigue, Emily Coffey, Chao Chen, Vedant, Carlo Pinciroli, Luca Giovanni Gianoli, and all those who walked through Polytechnique's MIST Lab for their guidance and expertise. I am grateful to Inoue-sensei, Nicolas Schwind, Maxime Clement, Tony Ribeiro, Morgan Magnin, David Martínez Martínez, Stefan Zeltner, and all the members of the Inoue Laboratory for what I was taught in Tokyo. Parts of this research would have not existed without Étienne Bourbeau, Gontran Sion, Marcello Valdatta, and the people of *PolyOrbite* and University of Bologna. I also learned a lot from Elburz Sorkhabi, Daniele Paglialunga, Gary Martin, and the staff and participants of ISU's Space Studies Program 2015, in Athens, Ohio.

Besides knowledge, I received plenty of help from the humans of Montréal (Pauline Grasset, Doris Adem, Zoé Arnaud, Nicolò Trambajolo, Nicola Cordoni, and Eddy Burgos), Tokyo (Cristiano Pedersini, Ivano Rotondo, Hannah So, Jo Spencer, Elissa Villa, Saki Alex Munehira, and Go Takei), Milan (Francesco Costa, Francesco Marconi, Riccardo Cattaneo, Matteo Carminati, Roberto Mangano, Francesco De Liva, and Marco Domenico Santambrogio), the *Club d'Athlétisme de l'Université de Montréal* (Florence Charbonneau-Dufresne, Éric Janvier, Franco Di Battista, Pierre-Antoine Dupont, Christophe Seiichi Martin, Gabriele Gori, and Clément Hely), my childhood (Marco Milanesi, Matteo Ottoboni, Alessio Elmi, Luca Cardinale, Roberto Bottini, and Matteo Floccari), and my family (Elena, Fabio, Alice, Selvio, Emilia, and Radio).

Thanks to the people who unwittingly led me into a PhD program—Alessandro Panella, Alessandro Gnoli, and Victoria Meissner—and to Stefania—for all the pity and understanding. Thanks to the CEPSUM, too. Ultimately, I am grateful to everyone who, through dialogue or example, contributed to shape this work (and me) over the course of the 2012–2017 period.

Finally, a special recognition goes to Antoine Morin, Guillaume Boglioni-Beaulieu, and Jean-Michel Tristan Robichaud for reviewing the *résumé* you will find on the next page.

# RÉSUMÉ

En raison de leur complexité croissante, les systèmes informatiques modernes nécessitent de nouvelles méthodologies permettant d'automatiser leur conception et d'améliorer leurs performances. L'espace, en particulier, constitue un environnement très défavorable au maintien de la performance de ces systèmes : sans protection des rayonnements ionisants et des particules, l'électronique basée sur CMOS peut subir des erreurs transitoires, une dégradation des performances et une usure accélérée causant ultimement une défaillance du système. Les approches traditionnellement adoptées pour garantir la fiabilité du système et prolonger sa durée de vie sont basées sur la redondance, généralement établie durant la conception. En revanche, ces solutions sont coûteuses et parfois inefficaces, puisqu'elles augmentent la taille et la complexité du système, l'exposant à des risques plus élevés de surchauffe et d'erreurs. Les conséquences de ces limites sont d'autant plus importantes lorsqu'elles s'appliquent aux systèmes critiques (e.g., contraintes par le temps ou dont l'accès est limité) qui doivent être en mesure de prendre des décisions sans intervention humaine. Sur la base de ces besoins et limites, le développement en aérospatial de systèmes informatiques avec capacités adaptatives peut être considéré comme la solution la plus appropriée pour les dispositifs intégrés à haute performance.

L'informatique auto-adaptative offre un potentiel sans égal pour assurer la création d'une génération d'ordinateurs plus intelligents et fiables. Qui plus est, elle répond aux besoins modernes de concevoir et programmer des systèmes informatiques capables de répondre à des objectifs en conflit. En nous inspirant des domaines de l'intelligence artificielle et des systèmes reconfigurables, nous aspirons à développer des systèmes informatiques auto-adaptatifs pour l'aérospatiale qui répondent aux enjeux et besoins actuels. Notre objectif est d'améliorer l'efficacité de ces systèmes, leur tolérance aux pannes et leur capacité de calcul.

Afin d'atteindre cet objectif, une analyse expérimentale et comparative des algorithmes les plus populaires pour l'exploration multi-objectifs de l'espace de conception est d'abord effectuée. Les algorithmes ont été recueillis suite à une revue de la plus récente littérature et comprennent des méthodes heuristiques, évolutives et statistiques. L'analyse et la comparaison de ceux-ci permettent de cerner les forces et limites de chacun et d'ainsi définir des lignes directrices favorisant un choix optimal d'algorithmes d'exploration.

Pour la création d'un système d'optimisation autonome—permettant le compromis entre plusieurs objectifs—nous exploitons les capacités des modèles graphiques probabilistes. Nous introduisons une méthodologie basée sur les modèles de Markov cachés dynamiques, laquelle

permet d'équilibrer la disponibilité et la durée de vie d'un système multiprocesseur. Ceci est obtenu en estimant l'occurrence des erreurs permanentes parmi les erreurs transitoires et en migrant dynamiquement le calcul sur les ressources supplémentaires en cas de défaillance. La nature dynamique du modèle rend celui-ci adaptable à différents profils de mission et taux d'erreur. Les résultats montrent que nous sommes en mesure de prolonger la durée de vie du système tout en conservant une disponibilité proche du cas idéal.

En raison des contraintes de temps rigoureuses imposées par les systèmes aérospatiaux, nous étudions aussi l'optimisation de la tolérance aux pannes en présence d'exigences d'exécution en temps réel. Nous proposons une méthodologie pour améliorer la fiabilité du calcul en présence d'erreurs transitoires pour les tâches en temps réel d'un système multiprocesseur homogène avec des capacités de réglage de tension et de fréquence. Dans ce cadre, nous définissons un nouveau compromis probabiliste entre la consommation d'énergie et la tolérance aux erreurs.

Comme nous reconnaissons que la résilience est une propriété d'intérêt omniprésente (par exemple, pour la conception et l'analyse de systèmes complexes génériques), nous adaptons une définition formelle de celle-ci à un cadre probabiliste dérivé à nouveau de modèles de Markov cachés. Ce cadre nous permet de modéliser de façon réaliste l'évolution stochastique et l'observabilité partielle des phénomènes du monde réel. Nous proposons un algorithme permettant le calcul exact efficace de l'étape essentielle d'inférence laquelle est requise pour vérifier des propriétés génériques. Pour démontrer la flexibilité de cette approche, nous la validons, entre autres, dans le contexte d'un système informatisé reconfigurable pour l'aérospatiale.

Enfin, nous étendons la portée de nos recherches vers la robotique et les systèmes multi-agents, deux sujets dont la popularité est croissante en exploration spatiale. Nous abordons le problème de l'évaluation et de l'entretien de la connectivité dans le contexte distribué et auto-adaptatif de la robotique en essaim. Nous examinons les limites des solutions existantes et proposons une nouvelle méthodologie pour créer des géométries complexes connectées gérant plusieurs tâches simultanément.

Des contributions additionnelles dans plusieurs domaines sont résumées dans les annexes, nommément : (i) la conception de CubeSats, (ii) la modélisation des rayonnements spatiaux pour l'injection d'erreur dans FPGA et (iii) l'analyse temporelle probabiliste pour les systèmes en temps réel. À notre avis, cette recherche constitue un tremplin utile vers la création d'une nouvelle génération de systèmes informatiques qui exécutent leurs tâches d'une façon autonome et fiable, favorisant une exploration spatiale plus simple et moins coûteuse.

# ABSTRACT

Today's computer systems are growing more and more complex at a pace that requires the development of novel and more effective methodologies to automate their design. Space, in particular, represents a challenging environment: without protection from ionizing and particle radiation, CMOS-based electronics are subject to transients faults, performance degradation, accelerated wear, and, ultimately, system failure. Traditional approaches adopted to guarantee reliability and extended lifetime are based on redundancy that is established at design-time. These solutions are expensive and sometimes inefficient, as they increase the complexity and size of a system, exposing it to higher risks of overheating and incurring in radiation-induced errors. Moreover, critical systems—e.g., time-constrained ones and those where access is limited—must be able to cope with pivotal situations without relying on human intervention. Hence, the emerging interest in computer systems with adaptive capabilities as the most suitable solution for novel high-performance embedded devices for aerospace.

Self-adaptive computing carries unmatched potential and great promises for the creation of a new generation of smart, more reliable computers, and it addresses the challenge of designing and programming modern and future computer systems that must meet conflicting goals. Drawing from the fields of artificial intelligence and reconfigurable systems, we aim at developing self-adaptive computer systems for aerospace. Our goal is to improve their efficiency, fault-tolerance, and computational capabilities.

The first step in this research is the experimental analysis of the most popular multi-objective design-space exploration algorithms for high-level design. These algorithms were collected from the recent literature and include heuristic, evolutionary, and statistical methods. Their comparison provides insights that we use to define guidelines for the choice of the most appropriate optimization algorithms, given the features of the design space.

For the creation of a self-managing optimization framework—enabling the adaptive trade-off of multiple objectives—we leverage the tools of probabilistic graphical models. We introduce a mechanism based on dynamic hidden Markov models that balances the availability and lifetime of multiprocessor systems. This is achieved by estimating the occurrence of permanent faults amid transient faults, and by dynamically migrating the computation on excess resources, when failure occurs. The dynamic nature of the model makes it adjustable to different mission profiles and fault rates. The results show that we are able to lead systems to extended lifetimes, while keeping their availability close to ideal.

On account of the stringent timing constraints imposed by aerospace systems, we then investigate the optimization of fault-tolerance under real-time requirements. We propose a methodology to improve the reliability of computation in the presence of transient errors when considering the mapping of real-time tasks on a homogeneous multiprocessor system with voltage and frequency scaling capabilities. In this framework, we take advantage of probability theory to define a novel trade-off between power consumption and fault-tolerance.

As we recognize that resilience is a pervasive property of interest (e.g., for the design and analysis of generic complex systems), we adapt a formal definition of it to one more probabilistic framework derived from hidden Markov models. This allows us to realistically model the stochastic evolution and partial observability of complex real-world environments. Within this framework, we propose an efficient algorithm for the exact computation of the essential inference step required to construct generic property checking. To demonstrate the flexibility of this approach, we validate it in the context, among others, of a self-aware, reconfigurable computing system for aerospace.

Finally, we move the scope of our research towards robotics and multi-agent systems: a topic of thriving popularity for space exploration. We tackle the problem of connectivity assessment and maintenance in the distributed and self-adaptive context of swarm robotics. We review the limitations of existing solutions and propose a novel methodology to create connected complex geometries for multiple task coverage.

Additional contributions in the areas of (i) CubeSat design, (ii) the modelling of space radiation for FPGA fault-injection, and (iii) probabilistic timing analysis for real-time systems are summarized in the appendices. In the author's opinion, this research provides a number of useful stepping stones for the creation of a new generation of computing systems that autonomously—and reliably—perform their tasks for longer periods of time, fostering simpler and cheaper space exploration.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| ADCS | Attitude Determination and Control System |
| ADRS | Average Distance from Reference Set |
| API | Application Programming Interface |
| ANN | Artificial Neural Network |
| APRS | Adaptive Windows Pareto Random Search |
| ASPPM | ArduSat Payload Processor Module |
| BMU | Behaviour Management Unit |
| CBBA | Consensus-Based Bundle Algorithm |
| CBS | Constraint-based System |
| CDF | Cumulative Distribution Function |
| CDR | Critical Design Review |
| C&DH | Command & Data-handling |
| CJPD | Conditional Joint Probability Distribution |
| CMOS | Complementary Metal-oxide-semiconductor |
| CSA | Canadian Space Agency |
| CSDC | Canadian Satellite Design Challenge |
| COTS | Commercial Off-the-shelf |
| DBN | Dynamic Bayesian Network |
| D-HMM | Dynamic Hidden Markov Model |
| DoE | Design of Experiments |
| DSE | Design Space Exploration |
| DVFS | Dynamic Voltage and Frequency Scaling |
| EA | Evolutionary Algorithm |
| EAC | European Astronaut Centre |
| ESA | European Space Agency |
| FFT | Fast Fourier Transform |
| FPGA | Field-programmable Gate Array |
| GA | Genetic Algorithm |
| HEO | Highly Elliptical Orbit |
| HMM | Hidden Markov Model |
| ILP | Integer Linear Programming |
| IMMOGLS | Ishibuchi-Murata Multiple Objective Genetic Local Search |
| IoT | Internet of Things |

| | |
|---|---|
| JPD | Joint Probability Distribution |
| LEO | Low Earth Orbit |
| LP | Linear Programming |
| MDP | Markov Decision Process |
| MILP | Mixed-Integer Linear Programming |
| MOGLS | Multiple Objective Genetic Local Search |
| MOMDP | Multi-objective Markov Decision Process |
| MOMSLS | Multi-Objective Multiple Start Local Search |
| MOO | Multi-objective Optimization |
| MOPSO | Multi-Objective Particle Swarm Optimization |
| MOSA | Multi-Objective Simulated Annealing |
| MPSoC | Multi-processor System-on-chip |
| MRTA | Multi-robot Task Allocation |
| MTTF | Mean Time To Failure |
| NBTI | Negative Bias Temperature Instability |
| NN | Neural Network |
| NSGA | Non-Dominated Sorting Genetic Algorithm |
| NSGAII | Controlled Non-Dominated Sorting Genetic Algorithm |
| OBC | On-board Computer |
| PE | Processing Element |
| PGM | Probabilistic Graphical Model |
| PI | Power Iteration |
| PMA | Pareto Memetic Algorithm |
| PMF | Probability Mass Function |
| POMDP | Partially Observable Markov Decision Process |
| PSA | Pareto Simulated Annealing |
| PSO | Particle Swarm Optimization |
| pWCET | Probabilistic Worst-case Execution Time |
| RESPIR | Response Surface Pareto Iterative Refinement |
| RFT | Reconfigurable Fault Tolerance |
| RNC | Robot Navigation Controller |
| RT | Real-time |
| RSM | Response Surface Modelling |
| SA | Simulated Annealing |
| SC | Swarm Centroid |
| SECS | Star Eruption for Connected Swarms |

| | |
|---|---|
| SEE | Single Event Effect |
| SEU | Single Event Upset |
| SGT | Spectral Graph Theory |
| SMOSA | Serafini's Multi-Objective Simulated Annealing |
| SPEA | Strength Pareto Evolutionary Algorithm |
| SPTA | Static Probabilistic Timing Analysis |
| TCTM | Total Completion Time Minimization |
| TDDB | Time-dependent Dielectric Breakdown |
| TDSM | Total Dislocation Space Minimization |
| TID | Total Ionizing Dose |
| TMR | Triple Modular Redundancy |
| TRL | Technology Readiness Level |
| TSG | Task Schedule Generator |
| VLSI | Very-large-scale Integration |
| WCET | Worst-case Execution Time |
| WP | Work Package |

# LIST OF APPENDICES

# CHAPTER 1 INTRODUCTION

"Science must begin with myths, and with the criticism of myths."
Sir Karl Raimund Popper, *Conjectures and Refutations: The Growth of Scientific Knowledge*, 1963

"Life can only be understood backwards; but it must be lived forwards."
"Det er ganske sandt, hvad Philosophien siger, at Livet maa forstaaes baglænds.
Men derover glemmer man den anden Sætning, at det maa leves forlænds."
Søren Kierkegaard, *Journalen*, 1843

This thesis has been written in partial fulfillment of the requirements for the degree of Philosophiæ Doctor in computer engineering. It recollects research work that was conducted within the MIST Laboratory of Polytechnique Montréal (Montréal, Québec, Canada) and the Inoue Laboratory of the National Institute of Informatics (Tokyo, Japan) between September 2012 and January 2017. The structure of the document is that of a thesis by articles—five published and submitted contributions are presented in the Chapters from 4 to 8.

## 1.1 Context and Motivation

Outside the protective cocoon of Earth's atmosphere, sharp changes in temperature, vacuum conditions, and a high level of radiation create an extremely harsh environment for digital electronics [63]. Without an atmosphere to protect from ionizing and particle radiation, CMOS-based computing systems are subject to transients faults, generalized performance degradation, accelerated wear, and, ultimately, system failure [100]. The design of space-grade computing hardware—often referred to as hardened components—requires large investments and it usually relies on government or military projects. The recent opening of the space sector to private enterprises [61], however, is creating a growing interest around for the use of low-cost, off-the-shelf components (COTS).

Adaptive and self-adaptive computing systems can reproduce the fault tolerance of hardened components through smarter algorithms and design rather than special production and validation processes. They allow to re-use COTS as their essential building blocks, saving money and time during the design and manufacturing phases. Enabling the use of COTS also opens the door to a range of new computationally intensive applications (e.g., nano-satellite Earth observation). Furthermore, a majority of the redundancy schemes currently used to improve fault tolerance—e.g., triple modular redundancy (TRM)—is expensive, sometimes wasteful,

and impractical for adoption in the soaring small- and nano-satellites (e.g., CubeSats) sector. These systems, in fact, often do not dispose of sufficient levels of replication and have to rely solely on software redundancy instead.

Aerospace computing systems are also characterized by stringent real-time requirements—similarly to all computing system whose correct functioning is life- or mission-critical [24]. Their software tasks have to execute (and terminate correctly) within fixed and predictable delays to ensure the schedulability (on a given hardware, a task set is said to be schedulable if an ordering of the tasks, such that all deadlines are met, exists) of the entire system. Failing to meet a deadline, in a small satellite, could lead to the attitude determination and control system (ADCS) neglecting to properly orient the solar panels, all energy being drained from the batteries and the loss of the spacecraft; in a manned spacecraft, to loss of life.

The scheduling of real-time systems is a heavily researched area at the intersection of mathematics and computer engineering [35]. Schedulability tests—inequalities revealing whether a certain set of tasks can execute before their deadlines—have been devised for a plethora of computer architectures [92, 106]. However, these tests typically rely on timing estimates—i.e., worst case executions times (WCETs)—that are extremely conservative, especially when it come to faults [28]. The drawback of inaccurate timing estimates is that systems are poorly exploited. This inefficiency leads to wasted computational power and more expensive designs. The promise carried by adaptive methodologies is that of autonomously adjusting the estimation of WCETs (in ways consistent to the changing fault rates of different mission profiles) to fully exploit the computational capabilities of a spacecraft. The growing sensitivity of airborne and ground-based electronics to the effects of radiation [128] (due to technology scaling) also entails that all fault mitigation strategies conceived for space critical systems will be in great demand in the near future.

The development of adaptive computing also addresses the challenge of programming computer systems that must meet conflicting goals, e.g., throughput, energy consumption, and reliability [120, 162]. Adaptive computer systems, in fact, can self-manage their resources to automatically find the best way to accomplish their goal, even under changing environmental conditions [114, 122]. The ability to autonomously adapt to the surrounding environment is a major asset for a computing system in space, where access to the hardware is limited or impossible. An adaptive system capable to detect and work around failures can optimize its behaviour and improve its performance accordingly. Nonetheless, adaptive computing's potential goes way beyond aerospace: results in this field can profit several other sectors, from industrial automation to personalized computing (e.g., virtual assistants).

Abstracting and developing this idea of mutating systems and environments to the utter-

most, one can see what the ultimate objective of adaptive computing is the replications, through engineering, of those natural properties that we often see expressed by resilient ecosystems. These are complex ecological systems having the particular ability to adapt to sudden, unpredictable changes [67, 169]. For example, a resilient computing system for aerospace could re-plan its mission phases—or even its mission objectives, discarding the less promising ones—once a major failure occurs. Resilient computing systems, however, are still a largely unexplored research subject. Hitherto, the scientific community has not yet agreed on formal definition of resilience.

Finally, a research work on adaptive computing system for aerospace cannot ignore the growing importance of robotics for space exploration. Robotic expeditions are the necessary first steps paving the way of the human exploration of the Solar System. Even after 40 years of robotic lander exploration (from Viking 1 to Philæ) the level of autonomous control on these machines is far from perfect [82]. This entails that, the further they go, the more inefficient they become (because of the round-trip delay necessary to provide human input). More autonomy is the gateway to fewer down-times waiting for control and greater scientific return. Besides probes and landers, autonomous control has the potential to increase safety and performance of launch vehicles too. Public and private companies such as ATK and SpaceX have plans to replace humans with algorithms in their flight termination systems.

Swarm robotics [21] has a strong appeal as an approach to implement the autonomous pursue of complex tasks using simple devices—and leveraging their number. Nonetheless, swam robotics is a very young discipline, confronting researcher with many new challenges. For example, how to maintain network connectivity across multiple robots moving in unknown environments where global positioning is not available (e.g., another planet) is still an open problem.

## 1.2 Problem Statement

The research work in this dissertation focuses on the investigation, definition, and implementation of formal methodologies for the modelling and design of adaptive computing systems for aerospace. In particular, its goal is the introduction of techniques to solve—or mitigate—the following challenges:

— The uncertainty surrounding the choice of the most appropriate algorithms (among the innumerable methodologies proposed in the literature) for the automated optimization of an embedded system design.

— The inefficiencies—in terms of computing resources utilization, power consumption,

and fault tolerance—of traditional redundancy schemes used in aerospace.

— The inefficiencies induced by excessively conservative WCET estimates in real-time computing systems for aerospace (especially those unaware of faults).

— The lack of a formal definition of what a resilient system is and what resilience entails.

— The lack of formal models and methodologies to assess or quantify resilience.

— The nonexistence of a distributed and resilient methodology to preserve network connectivity in a multi-robot system exploring an unknown environment.

In the opinion of the author, addressing and resolving these problems has the potential to substantially advance of the frontier of knowledge in the fields of computing systems for aerospace and multi-robot system design.

### 1.3 Research Objectives

The challenges outlined in the previous section are addressed—through the research articles presented in the Chapters from 4 to 8—by pursuing the following objectives:

1. Conceive a formal method to classify and compare the existing methodologies for the design-space exploration of an embedded system.
2. Introduce rigorous and realistic tools to model those aspects of the space environment that affect the operations of a computing system.
3. Using these tools, discover new, non-obvious relationships between performance metrics such as energy consumption, real-time execution, and fault tolerance.
4. Model aerospace computing systems through an intelligent framework that allows to answer (probabilistic) inference queries—in particular with regard to the system's resilience.
5. Exploit this knowledge to: (i) implement adaptive fault tolerance and (ii) investigate high-level properties of complex systems such as resilience and connectivity.
6. Implement an adaptive and distributed algorithm that enforces connectivity in a multi-robot system.

Furthermore, Appendix A summarizes the efforts made towards the (less research- and more engineering-oriented) objective of developing the on-board computing systems of the 3U CubeSats of Polytechnique Montréal's student society *PolyOrbite*.

## 1.4 Novelty and Impact

To the best of our knowledge, the major aspects of novelty of the research presented in this dissertation are represented by the following contributions:

— A 4-class taxonomy of design-space exploration strategies for the optimization of embedded computing systems—as well as their cross-class experimental comparison.

— A classification methodology—based on probabilistic graphical models—to distinguish between transient and permanent faults induced in CMOS circuits by space radiation.

— A trade-off between energy consumption and fault-tolerance exploiting the utilization levels and the dynamic voltage and frequency scaling capabilities of homogeneous multiprocessor systems for the scheduling of hard real-time task sets.

— A formal definition of resilience in the context of probabilistic graphical models; the identification of an efficient algorithm to perform inference on partially-observed time series; and the study of its complexity and implications on generic property checking.

— A partially distributed approach for the simultaneous spatial coverage of multiple tasks by a swarm of robots that preserves, at the same time, the team connectivity.

The scientific significance and potential impact of the investigation include:

— A set of guidelines for the choice of a design-space exploration algorithm that will help both the designers of computer hardware and the developers of the automation tools.

— Adaptive strategies able to improve how redundancy is exploited by fault-tolerant aerospace computing systems and, overall, to make their design cheaper and faster.

— Better autonomous multi-robot systems that are especially suitable for deployment in unknown and harsh environments, e.g., for disaster response and space exploration.

— Finally, the tangential—but not negligible—educational repercussions obtained in the context of Polytechnique Montréal and Canada through *PolyOrbite* (Appendix A).

The relevance of these themes is current news and it can only be expected to grow. At the time of the writing, for example, the hostile and high radiation environment of Fukushima I's damaged nuclear power plant—one of the consequences of 2011's Tōhoku earthquake and tsunami—is continuing to fatally challenge the robots that try to assess and contain the radioactive leak [1]. Multi-robot systems that are (i) autonomous, (ii) fault tolerant, (iii)

1. `http://www.japantimes.co.jp/news/2017/02/17/national/fukushima-fuel-removal-quest-leaves-trail-dead-robots/#.WL8Y_BIrKYV`

resilient to radiation and harsh environments, and (iv) self-organizing are a *sine qua non* for the advancement of space exploration as well as crisis resolution here on Earth.

Focused on the space domain, this research has the potential of improving efficiency, fault tolerance, and computational capabilities of aerospace computing and robotic systems—as well as to benefit a plethora of tomorrow's ground-based systems. In the author's opinion, this research set useful stepping stones for the creation of a new generation of computing systems that are able to autonomously perform their tasks for longer periods of time, fostering simpler and cheaper space exploration.

# CHAPTER 2 LITERATURE REVIEW

"Every man takes the limits of his own field of vision for the limits of the world."
"Jeder hält das Ende seines Gesichtskreises für das der Welt."
Arthur Schopenhauer, *Parerga and Paralipomena, Psychologische Bemerkungen*, 1851

"Those who do not remember the past are condemned to repeat it."
George Santayana, *The Life of Reason, Vol. I, Reason in Common Sense*, 1905

This chapter is dedicated to the revision of fundamental concepts and existing research contributions in four different areas: hardware-software co-design and optimization; adaptive hardware and software; the resilience of complex systems; and multi-agent systems and swarm robotics. All of these themes are equally relevant for this research—ideally placed at the centre of the 4-way intersection they create. Not to weigh down the discussion with unnecessary repetition, cross references to the sections of each chapter that cover similar topics are also provided. The recapitulatory Table 2.1 concludes the chapter.

## 2.1 Hardware-software Co-design and Optimization

The challenge of designing embedded computing systems, and MPSoC in particular, was presented and studied in [99]. Complications primarily arise from the large number of parameters at play and the complex—sometimes unknown—relationships between components and the several performance metrics under scrutiny. Therefore, multi-objective optimization emerges as a framework with significant appeal for the automation of this design process. Because of the profusion of methodologies in the area, however, it is not always easy to choose the right tools for any given problem. Several surveys and comparative studies aim at facilitating the decision making of researchers and designers. A survey of multi-objective optimization algorithms—not specific to DSE—by Marler and Arora [98]—reviewed multiple approaches, including genetic algorithms, to find that no single approach could be declared superior to all others in the general case. The survey of Marler and Arora [98] distinguished three types of multi-objective optimization algorithms: "methods with a priori articulation of preferences, methods with a posteriori articulation of preferences, and methods with no articulation of preferences". Approaches in the last category are the most flexible and suitable for automation (and popular in DSE), however, they add to the complexity of the problem. Rather than a single answer, they output a set (usually called Pareto set/front) of candidate

solutions that meet the requirements of Pareto optimality [26]. As a consequence, best-effort algorithms—that do not guarantee to return the set with all-and-only the Pareto optimal solutions but only produce an approximate Pareto front—require additional methodologies to assess the quality of their results. The work of Taghavi and Pimentel [161] is a comprehensive review of metrics and visualization techniques for the performance of multi-objective evolutionary algorithms.

This additional layer of complexity is especially important—and usually unavoidable—in the context of DSE. The design-space of an embedded computing systems is often unsearchably large, making the use of approximate optimization a necessity. Consequently, evolutionary and genetic algorithms—as well as other heuristic strategies—are extremely popular in the field. Coello reviewed the performance of several approaches based on genetic algorithms for multi-objective optimization in [30] and [31]. Fonseca and Fleming [48] and Zitzler *et al.* [181, 180] reviewed and compared multi-objective evolutionary algorithms. They point out, in particular, the importance of elitism—i.e., the mechanisms used to preserve the best solutions in the population of a GA across evolutionary iterations. The work in [77] includes a comparative review of meta-heuristics for a two-objective set covering problem, while multi-objective combinatorial optimization algorithms are surveyed in [166]. Not all DSE approaches, however, are based on heuristics and pseudo-random searches. Hegedüs *et al.* [62], for example, proposed a model-based DSE approach that uses dependency analysis and the algebraic abstraction of transformation rules to generate the following solution to evaluate. Orthogonally to all of these approaches, Shao *et al.* [151] recently demonstrated that the performance of any design-space exploration strategy for hardware accelerators can be greatly improved using a more accurate pre-RTL simulator.

The contribution of this dissertation provides for two significant deficiencies in the existing research: (i) the fact that most comparative works limit their scope to only certain families of algorithms; and (ii) the lack of a practical but generic protocol for the choice of a MOO algorithm in a given DSE problem. Chapter 4, in fact, presents a quantitative comparison of DSE algorithms from multiple and different theoretical backgrounds (including EAs and GAs) as well as guidelines for their application. Since its publication in [116], it has already contributed to help designers and researchers. Alouani *et al.* [5] and Mediouni *et al.* [101], for example, based their approaches on the IMMOGLS algorithm because of the results in [116]. Additional information on DSE surveys and comparative research can be found in Section 4.2 while Section 4.4 summarizes 15 of the most popular design-space exploration strategies.

## 2.2 Adaptive Hardware and Software

Adaptive capabilities can benefit a wide range of applications—from search algorithms [85] to mechatronics [1]. The hardware and software of aerospace systems require the ability to adapt because of the harsh, dynamic environment they operate in. The presence of intense radiation in space, in fact, is responsible for both hard failures and the onset of a wide range of soft errors deriving from single event effects (and upsets) [128, 80]. Designing adaptive systems that can work around these faults is a necessity for aerospace applications and, prospectively, for ground-level safety-critical systems too: as pointed out by Alexandrescu *et al.* [4], shrinking technology processes are making all types of circuits more and more sensitive to radiation. The work in [4] proposes a fault injection framework for the assessment of this sensitivity. Most of the literature in the area can be partitioned into two groups: (i) research that deals with the detection and recovery from permanent failures; and (ii) research that deals with the correction and mitigation of transient errors. The first category includes: [60], an efficient detection method for hard faults that uses an adaptive scrubbing period and it was tested on a 16-core multiprocessor; [156], a wear-out fault detector characterized by minimal hardware overhead and infrequent periodic monitoring; [25], a software flow for both the detection and correction of permanent faults arising in SRAM FPGAs—Cassano *et al.* also point out the "need to include the soft error rate (SER) as another design parameter"; and [103], an optimization method that protects redundant systems from permanent faults through the cost-effective allocation of slack resources. The framework proposed by Jacobs *et al.* [75] focusses, instead, on transient errors. It comprises "an adaptive hardware architecture" and "an upset rate modeling tool that captures time-varying radiation effects for arbitrary satellite orbits" (similarly to the work mentioned in Appendix B). However, permanent failures are not accounted for. The work Chapter 5 distinguish itself as a more holistic approach in that it assumes that both permanent and transient faults can occur and it offers a methodology to tell them apart (see also Table 2.1).

The research work in Chapter 6 defines an adaptive trade-off between energy consumption and fault tolerance for multiprocessor systems, taking into account the real-time requirements and peculiarities of the space environment. The computing systems under study are characterized by identical PEs because, as observed by Davis and Burns [35], a majority of the results on real-time scheduling for multicores has been derived on homogeneous architectures. The most resembling research work in the literature is that of Bolchini *et al.* in [19]—an operating system module for adaptive reliability against permanent and transient faults in many-core systems—and [20]—a run-time approach for energy/performance trade-off in many-core systems. However, it should be observed that, in [19], the fault model is

not space-specific nor formally grounded into statistics and the decision/inference layer is rather scant. Furthermore, the approach in [20] was not intended for real-time applications. Additional information on these subjects can be found in Sections 5.2 and 6.2.

Some of the most recent and significant contributions in the context of adaptive computing systems also include: ReconOS [2], an operating system enabling reconfigurable computing through a unified multithreaded programming model and OS services for those threads that are mapped to reconfigurable hardware; and [29], an adaptive cache management strategy for energy-efficient GPU computing that exploits both prediction and run-time detection of contention. The work of [2], in particular, substantiates the choice of developing adaptivity at the operating system level, as suggested in Chapter 5 and 6.

## 2.3 Resilience of Complex Systems

Holling [67] originally defined resilience as the characterizing property—opposed to stability—of those ecological system that are capable of enduring great transformations while remaining functional. More recently, these ideas were extended by the same Holling [68] and Walker *et al.* [169] with the concepts of adaptability, transformability and sustainability of complex systems—sustainability, in particular, was defined as "the capacity to create [...] and maintain adaptive capability". Furthermore, the research in [68] and [169] highlighted that these properties apply not only to ecology but also to social and economical systems (this dissertation argues that computing and robotic systems should be added to the list). This shift—from ecology to multiple sciences—was noticed by Folke [47]: "history was dominated by empirical observations of ecosystem dynamics interpreted in mathematical models" that were used to develop an "adaptive management approach for responding to ecosystem change". Today, a "serious attempts to integrate the social dimension is [...] taking place in resilience work" and this is "reflected in the large numbers of sciences involved in explorative studies and new discoveries of linked social-ecological systems". For example, [47] points out the new interests in multi-agent systems and "adaptive governance". The interest in resilience has become so widespread that now it involves the insurance sector and financial regulators [39] who must develop metrics to assess, e.g., the resilience of specific industries to the occurrence of natural disasters. Stress tests to evaluate portfolios' risks, is found by [39], can bring "a deeper understanding of the relationship between hazard, exposure and vulnerability" and help assigning "a quantitative value on physical and operational resilience".

In the field of artificial intelligence, Schwind *et al.* in [148] and [149] proposed a definition of resilience comprising of three other sub-properties, namely: resistance, functionality, and recoverability. This definition was originally applied to dynamic and constraint-based systems.

The work in Chapter 7 builds upon [148] in two successive steps: (i) it points out the limits in descriptive power of the approach in [149] and (ii) it enriches it with stochastic evolution and partial observability. This is achieved exploiting the frameworks of probabilistic graphical models and, in particular, dynamic Bayesian networks [86]. The great expressive power of these models, however, comes at the cost of high computational complexity. This motivates the identification of (i) specific queries that can be answered exactly and efficiently (Chapter 7) and, in parallel, of (ii) techniques to improve the efficiency of approximate inference in generalized models [70]. Additional information on resilience and artificial intelligence can be found in Section 7.2.

## 2.4 Multi-agent Systems and Swarm Robotics

Swarm robotics is an emerging research area at the crossroad of mechanical engineering, biology, and artificial intelligence. Brambilla *et al.* [21] and Bayındır [13] reviewed its engineering challenges and potential applications (aggregation, flocking, foraging, path formation, collaborative manipulation task allocation, etc.). To address the swarm engineering problem from a computer scientist's perspective, Pinciroli *et al.* [132] introduced a set of development tools comprising of a swarm-specific programming language (Buzz) and APIs (e.g., for message passing and distributed consensus). Because a swarm of robots can be seen as a network of mobile nodes, the study of this field is also strongly intertwined with those of sensor networks and the IoT. The distributed assessment of a swarm's topology and its properties—connectivity in particular—is a prolific research area. Several publications deals with different facets of this problem (see also Table 2.1): [18] is a decentralized method to estimate topology changes (not necessarily global disconnections) exploiting the properties of synchronizing chaotic oscillators; [126] uses a similar methodology to discover symmetries within networks; [89] is a heuristic control strategy to maintain connectivity in a swarm of robots subject to contrasting forces; [17] and [37] are distributed algorithms for the computation of the second eigenvalue of the Laplacian matrix of a network—a gauge for connectivity—based on the power iteration method; [175] adds to the previous two works with the ability to distributedly estimate all eigenvalues and eigenvectors; [145] is an algorithm with the same objective of the previous two but based on the wave-equation propagation. Section 8.2 includes a review of the theoretical background of all these methodologies as well as a comparative re-implementation of those based on the spectral graph theory (i.e., computing eigenvectors and eigenvalues).

The approach outlined in Chapter 8 is a novel connectivity-aware distributed controller—implemented in Buzz [132]. Unlike the methodologies mentioned above, it exploits Lennard-

Jones potential forces to drive robots into organized structures while also preserving (not just assessing) their connectivity. The versatility of potential fields in the context of multi-robot systems, e.g., for collision avoidance, was previously acknowledged by Cruz *et al.* in [32]. Additional information on swarm robotics research and connectivity are given in Section 8.2.

More recently, Zelazo *et al.* [178] proposed a decentralized strategy for maintaining the formation rigidity of a multi-robot system (using range measurements) as its graph topology changes—a related but different problem with respect to that of connectivity. Antonelli *et al.* [6] introduced a distributed controller–observer model for the tracking and control of the centroid of a multi-robot system. Soltero *et al.* [158] and Yazici *et al.* [173] presented, respectively, a decentralized and an energy-aware methodology for path planning and task coverage. Unlike the simultaneous task coverage solution detailed in Chapter 8, [158] and [173] are inherently sequential approaches.

Advancements in hardware and software for autonomous robotics are essential to foster planetary and space exploration [82]. The work of Bajracharya *et al.* [9] outlines the progression in autonomous capabilities over three generations or NASA's and JPL's Mars landers: (i) Pathfinder and Sojourner; (ii) Spirit and Opportunity (Mars Exploration Rover); and Curiosity (Mars Science Laboratory). While the first generation of rovers was only provided with basic abilities such as terrain navigation and resource management; MER robots can perform "visual pose estimation" and "automatically detect science events". With regard to terrestrial applications and technology spin-outs, Apvrille *et al.* [7] also explained the importance of autonomous robots for "assisting rescue services within the context of natural disasters", underlining the role that drones can play in life-saving information relay. Finally, a growing interest is surrounding the study of resilience and resilient behaviours in the context of multi-robot systems [55]. For example, the work of Saldaña *et al.* [146]—in which resilience is described as the property of those robot networks that can reach consensus even in presence of faulty or malicious devices—proposes a resilient formation building algorithm.

Table 2.1 Comparison with similar research work in the literature. For each one of the main aspects of this research, a selection of comparable contributions is reported. A major differentiating feature is also listed besides each contribution.

| Feature | Related | Dissimilarities |
|---|---|---|
| A survey of multi-objective | [30, 31] | Genetic algorithms only. |
| optimization (MOO) algorithms | [180, 182] | Evolutionary algorithms only. |
| for DSE (Chapter 4). | [48] | Evolutionary algorithms only. |
| MOO taxonomy/guidelines(Ch.4). | [98] | Not specific to DSE. Non-Pareto MMO. |
| A model of the permanent | [60] | Transient error modelling is overlooked. |
| faults and wear-out induced | [156] | Transient error modelling is overlooked. |
| by TID (Chapter 5). | [25] | Transient error modelling is overlooked. |
| A model of the transient faults | [75] | Permanent faults are ignored. |
| due to space radiation (Ch.5). | [80] | Not an error mitigation approach. |
| A power/reliability trade-off | [19] | Not specific to space systems and faults. |
| for space RT systems (Chapter 6). | [20] | Not intended for real-time systems. |
| A formal definition of probabilistic | [169] | Informal definition of resilience. |
| resilience and the properties it | [148] | Non-probabilistic definition of resilience. |
| comprises (Chapter 7). | [146] | Applicable to networked systems only. |
| A distributed potential-based | [89] | Heuristic connectivity control. |
| robot controller to preserve | [17, 37, 175] | PI-based connectivity assessment. |
| the connectivity of a robot | [18] | Topology-, not connectivity-aware. |
| swarm (Chapter 8). | [145] | Wave eq.-based connectivity assessment. |
| Simultaneous task coverage for | [158] | Not energy-aware and sequential method. |
| swarm robotics (Chapter 8). | [173] | Sequential path planning method. |

# CHAPTER 3 RESEARCH APPROACH AND THESIS ORGANIZATION

"All human knowledge begins with intuitions,
proceeds from thence to concepts, and ends with ideas."
"So fängt denn alle menschliche Erkenntnis mit Anschauungen an, geht von da zu Begriffen, und endigt mit Ideen."
Immanuel Kant, *Kritik der reinen Vernunft*, 1781/1787

"Science may be described as the art of systematic over-simplification."
Sir Karl Raimund Popper, *The Open Universe: An Argument for Indeterminism*, 1982

This chapter presents how to address the problematics and pursue the objectives outlined in Chapter 1. The proposed methodology comprises five work packages (WPs) distributed across a three-layer development stack. Section 3.1 provides a condensed but comprehensive view of the approach. The subsequent section details the WPs in each layer. Finally, the overall structure of the document is described.

## 3.1 Bird's-eye View

Figure 3.1 offers an overview of the logical—and, for the most part, chronological—development of the research project presented in this dissertation. The background colours of the image frame the project's activities with respect to a minimalist computer engineering development stack comprising of three layers: the hardware level, the operating system (OS) level, and the application level. The five circles represent the work packages that led to the generation of this thesis' main contributions. The ordering of the WPs follows a logical bottom-up [1] approach:

1. WP1 includes a comparative analysis of the literature on design-space exploration for the optimization of embedded computing systems [116], i.e., the contribution presented in Chapter 4.
2. WP2 assembles the efforts made to understand, capture, and represent the faults induced by space radiation using probabilistic [115] and phenomenological models [167], i.e., the contributions in Chapter 5 and Appendix B.

1. Typically, the "bottom" refers to the simpler elements that are integrated on the way "up", towards a more complex system. In Figure 3.1's presentation, this vertical ordering is reversed to comply with the Gutenberg Diagram's design pattern [153].

3. The tools developed in WP2 were then used to investigate adaptive resource management at the operating system level in WP3 [115, 117]. These are the contributions in Chapter 5 and Chapter 6.
4. WP4 generalizes the concept of adaptiveness developed in WP3 using a similar framework for the assessment of the resilience of a system (Chapter 7).
5. Finally, WP5 evaluates adaptiveness and artificially enforced resilience through a distributed application: a multi-robot system that can cover multiple spatially distributed tasks without losing its connectivity (Chapter 8).

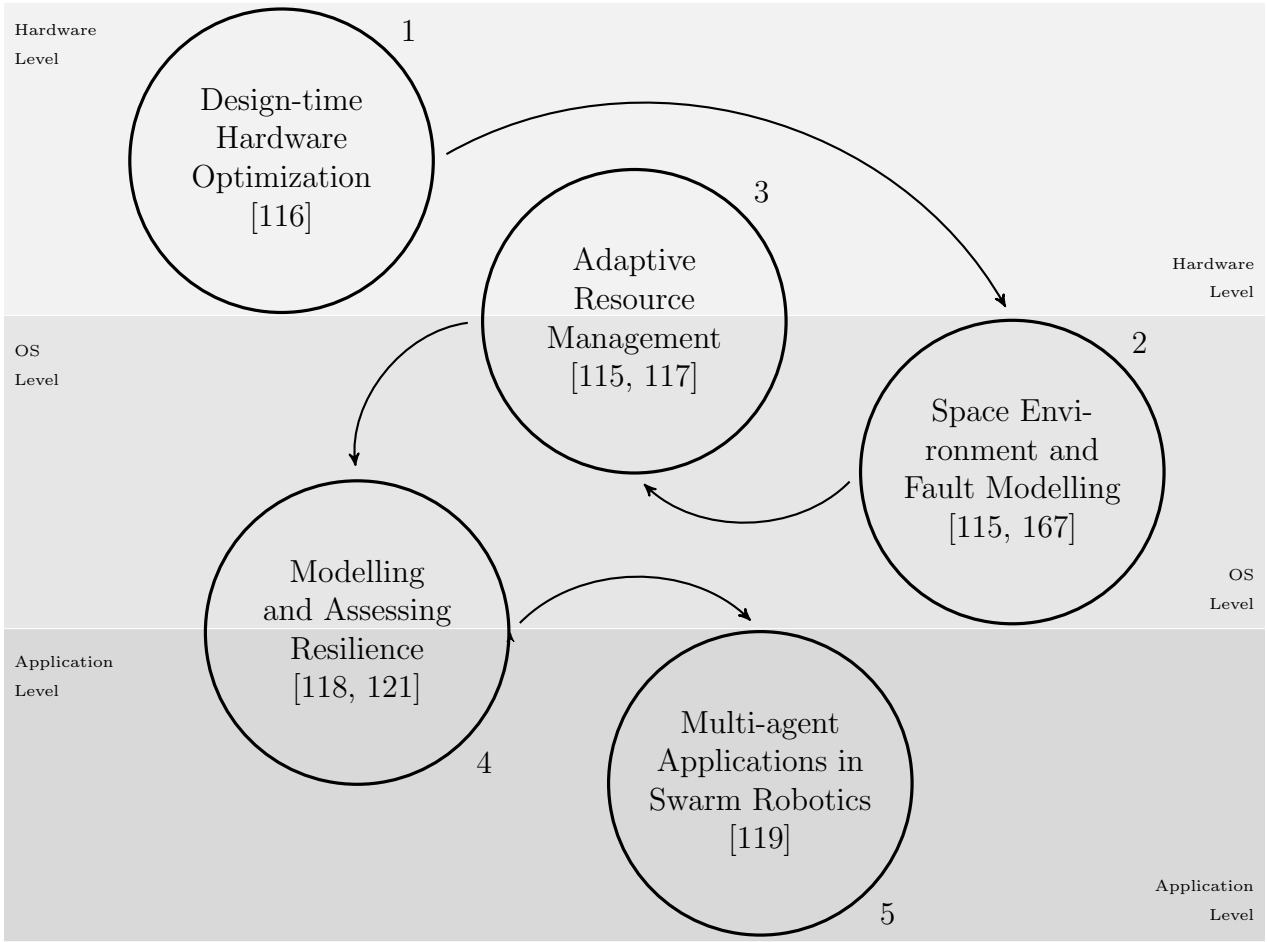



Figure 3.1 Overview of the main work packages in this research project, their logical (and chronological) ordering, and their position in the development stack.

Additional co-authored research efforts, not included in the five work packages of Figure 3.1 are summarized in Appendices A and C. These endeavours include the static probabilistic

timing analysis (SPTA) of the methodology described in [115] with Chao *et al.* [27], educational outreach, and CubeSat design [46].

## 3.2 Methodology

The approaches and methodologies adopted throughout the different phases and work packages of this research are outlined in the following subsections. Their presentation follows the three-layer paradigm of Figure 3.1—i.e., the hardware level is first, the operating system level second, and the application level last.

### 3.2.1 Hardware Design Level

WP1 starts this research at the lowest level of the development stack, i.e., the hardware design level. The reason of this bottom-up approach is the desire to mitigate the risk of failing to understand the nuances related to the low-level functioning of a computing system. The research output of WP1—a qualitatively and quantitatively comparative survey—was published in the ACM Transactions on Design Automation of Electronic Systems [116].

The analysis in [116] (Chapter 4) compares several multi-objective optimization algorithms for hardware design. Using the notion of Pareto optimality [26], it assesses the quality of the solution set produced by each algorithm through multiple performance metrics (e.g., ADRS and concentration) and a rigorous statistical study. The article also contains (i) a taxonomy proposal, (ii) a qualitative/semi-quantitative analysis of the ease of use of the algorithms, and (iii) suggested guidelines for their application.

**Libraries and Algorithmic Implementations** WP1 exploited the open source implementations of several search algorithms in the C++ Multiple Objective MetaHeuristics Library (MOMHLib++) of Jaszkiewicz and Dąbrowski [78], the work of Zaccaria *et al.* [174], Multicube Explorer (M3Explorer), and the original source code of the MDP and MOMDP algorithms [15, 16].

**RESP Simulation Environment and Benchmark Applications** The performance of the computing architectures (object of the optimization process) was estimated through ReSP [14], an open-source simulation environment. The benchmark application set comprised: *ffmpeg*, a video transcoder; *pigz*, a parallel compression algorithm; and *fft6*, Bailey's 6-step FFT algorithm.

### 3.2.2 Operating System Level

Having understood how one could pick the best components for an embedded computing system [116], the investigation moved up in the development stack (down-right in Figure 3.1) to work packages 2, 3, and 4. The operating system is the prime software layer for the management of the hardware resources of a computing device. This level of the stack is, perhaps, the most promising one for the development of adaptive behaviours [11].

The work authored and co-authored in this layer include the conference papers from WP3 [115, 117] (Chapters 5 and 6) and [27, 167] (see Appendices B and C). Their most relevant methodological features are:

1. The combination of probability theory, phenomenological models, and reconfigurable hardware [3] to model and simulate the effects of space radiation on a computing system.
2. The exploitation of timed probabilistic graphical models to describe the evolution of aerospace computing systems; and their combination with non-memoryless probability distributions to capture the cumulative effects of ionizing radiation.
3. The development of a quantitative model—inspired by the principles of VLSI design—to relate (i) the power consumption of a real-time multiprocessor system and (ii) its fault tolerance to transient errors *via* its utilization levels.

**CREME96 Model** The work in [167], [115], and [117] require knowledge of the average transient fault rates experienced by a spacecraft. The CREME96 tool suite [164], developed by NASA and Vanderbilt University, provides a web service [2] to satisfy this need. Once fed with the appropriate input parameters—i.e., the orbital state vectors, the L-values, and the parameters modelling an FPGA semiconductor material—CREME96 outputs a position-specific single event effect (SEE) rate. The results of [167] were validated against the experimental data recorded during MISSE-7 in a Virtex-4 device developed by NASA's Goddard Space Flight Center and deployed on the International Space Station [129].

**Probabilistic Graphical Models** The research presented in Chapters 5 and 7 heavily relies on the formal frameworks of probabilistic graphical models (PGMs) and dynamic Bayesian networks (DBNs) [86, 143]. As the (discrete) temporal extension of traditional Bayesian networks [86], DBNs are fully described by two components: (i) the "time 0 Bayesian network", i.e., the joint probability distribution over $N$ random variables at time 0; and (ii) the "2-time-slice Bayesian network" (2-TBN), a conditional BN defined over the $N$ variables

2. `https://creme.isde.vanderbilt.edu/`

at time $t$ and those at $t+1$. The 2-TBN only contains arcs from the variables at time $t$ to the variables at time $t+1$ (inter-time step influences), and arcs among the variables at time $t+1$ (intra-time step influences). Both the "time 0 Bayesian network" and the 2-TBN are BN themselves—i.e., directed acyclic graph in which each node is a random variable described by the conditional probability distribution with respect to all of the variables in its parent nodes. Hidden Markov models (HMMs) [143], in particular, are a specialized PGM that lies at the intersection of state-observation models and dynamic Bayesian networks.

A typical example of a stochastic system that can be efficiently described with a DBN is that of a robot navigating through an unknown environment using noisy sensors. The "time 0 BN" would represent the ground belief of the robot—as it activates—and the 2-TBN would be used to describe the evolution of the probability distribution of its sensor readings—as it moves.

#### Defining and Assessing the Resilience of Complex Systems

The research work in WP4—on resilience and the algorithmic means of quantifying it—lies at the crossroad between resource management (the OS level) and resource exploitation (the application level). Resilience is a property derived from ecology [67] and it strongly appeals to the designers of critical and aerospace systems because of the aura of unbreakability it emanates.

A framework based on hidden Markov models (enriched with a cost function) is exploited in Chapter 7 to formally describe resilience in the context of stochastic and partially-observable environments. The additional theory provided in [121] (Chapter 7) includes four application scenarios and the analytical and experimental complexity studies of an exact inference algorithm. The algorithm's MATLAB-compatible Octave implementation is available on GitHub [3].

### 3.2.3 Application Layer

Finally, the study raises to the application layer with the intention of leveraging the matured understanding through a demonstration of adaptive and resilient technology. To reflect the multifaceted research interest of MIST Laboratory in space and robotics, we chose to investigate the implementation of autonomous multi-robot exploration.

The work performed within this layer is that of WP5 and it is summarized by the article under review [119] in Chapter 8. The main methodological features of this study include (i) the

3. `https://github.com/JacopoPan/probabilistic-resilience.git`

review of a large body of literature in the field of connectivity assessment (and maintenance) for networked multi-robot systems, and (ii) the reproduction and critique of several results published in the area. Furthermore, we propose a novel two-level controller that circumvents the issues exposed by our review and leverages potential forces to enforce connectivity in the presence of conflicting goals (tasks). To allow the reproduction of our results, the Python Jupyter Notebook and MATLAB-compatible Octave implementations of this work are also available on GitHub [4] [5] [6] [7] [8].

**ARGoS Robotic Simulator** The results presented in Chapter 8 were obtained using the multi-physics robot simulator ARGoS [133]. ARGoS efficiently simulates large-scale swarms of robots and complex real-life interactions such as collisions, inertia, and communication failures (e.g., package drop).

**BUZZ – A Programming Language for Swarm Robotics** Albeit ARGoS supports robot controllers written in C++, the experiments of Chapter 8 use Buzz, a swarm-specific programming language [131] developed at MIST Laboratory. Buzz features built-in consensus mechanisms (i.e., virtual stigmergy [132]) and APIs to create swarms within swarms.

### 3.3 Document Structure

The document follows the traditional layout for dissertations by articles, that is, the inclusion of published or submitted contributions in the body of the work, each as a separate chapter.

— Chapter 1 is the introduction of the dissertation. It describes the research questions, objectives, and aspects of novelty.

— Chapter 2 reviews the state of the art in four related areas: (i) hardware-software co-design and optimization, (ii) adaptive hardware and software, (iii) fault-tolerance and resilience, and (iv) multi-agent systems and swarm robotics.

— Chapter 3 is this chapter. It contains an overview of the research approach and of the structure of the dissertation.

— Chapters from 4 to 8 are the published or submitted articles that compose the heart of the thesis.

4. `https://github.com/JacopoPan/ar-spectral-graph-theory-comparison.git`
5. `https://github.com/JacopoPan/ar-discrete-chaotic-oscillators.git`
6. `https://github.com/JacopoPan/ar-flocking-and-leader-forces.git`
7. `https://github.com/JacopoPan/ar-prominences-in-the-ideal-world.git`
8. `https://github.com/JacopoPan/ar-argos-buzz-simulations.git`

  - Chapter 4, is a comparative survey of the literature (part of WP1) published in the ACM Transactions on Design Automation of Electronic Systems in 2014 [116].
  - Chapter 5 is a conference paper (part of WP2 and WP3) presented at the 2014 NASA/ESA Conference on Adaptive Hardware and Systems in Leicester, United Kingdom [115].
  - Chapter 6 is a conference paper (part of WP3) presented at the 2015 NASA/ESA Conference on Adaptive Hardware and Systems in Montréal, Québec, Canada [117].
  - Chapter 7 is a submitted journal article (part of WP4 and a research internship at the National Institute of Informatics in Tokyo, Japan) under review for Science Advances.
  - Chapter 8 is a submitted journal article (part of WP5) currently under review for Autonomous Robots' Special Issue on Distributed Robotics: From Fundamentals to Applications.
- Chapter 9 discusses the results presented in the previous five chapter going through the same four areas of Chapter 2.
- Chapter 10 provides conclusions, lessons learned, unanswered questions and future work directions.
- The appendices summarize a selection of side project and co-authored work.
  - Appendix A reports on the engineering and educational efforts in CubeSat design of technical society *PolyOrbite*.
  - Appendix B recapitulates the contribution of a short paper (part of WP2) by Vedant *et al.* [167] on the modelling—and injection into FPGAs—of errors induced by space radiation.
  - Appendix C rehashes the significance of a conference paper by Chao *et al.* [27] on the static probabilistic timing analysis of a system equipped with the tools presented in [115].

The attentive reader will have observed that each of the "non-article" Chapters 1, 2, 3, 9, and 10 debuts with epigraphs from the work of famous thinkers and Western philosophers of the 18th, 19th and 20th century. The purpose of these quotations is to set the tone of each chapter as well as to reflect the ambiguous sense of humour of the author.

# CHAPTER 4 ARTICLE 1 – A COMPARATIVE EVALUATION OF MULTI-OBJECTIVE EXPLORATION ALGORITHMS FOR HIGH-LEVEL DESIGN

**Preface:** The starting point of this research is the threatening growth in complexity of aerospace on-board data-handling systems. A serious challenged posed by the design of modern embedded computing systems is the fact that the number of choices they entail—relative to their components and their parametrization—is so large that it is often impossible to evaluate all of them. In the article presented in this chapter, we systematically analyze the data that we collected experimenting with 15 search algorithms—gathered from the existing literature—to automate the design process. Choosing the most suited algorithm for this problem has the potential to improve a system's performance and, at the same time, to contain its design cost. Besides the raw comparisons in time to convergence and Pareto optimality, our findings suggest that pseudo-random approaches can often suffice in small search spaces but that the methodologies that perform best with difficult problems (i.e., large or hard to simulate search spaces) are those that exploit probabilistic and statistical models. These results are also the premise of a chapter in the "Handbook of Hardware/Software Codesign" edited by Soonhoi Ha and Jürgen Teich (to appear). We use the discovery to direct the rest of our research.

**Full Citation:** Jacopo Panerati and Giovanni Beltrame. 2014. A comparative evaluation of multi-objective exploration algorithms for high-level design. *ACM Transactions on Design Automation of Electronic Systems* 19, 2, Article 15 (March 2014), 22 pages.

**DOI:** `http://dx.doi.org/10.1145/2566669`



## Abstract

This paper presents a detailed overview and the experimental comparison of 15 multi-objective Design Space Exploration (DSE) algorithms for high-level design. These algorithms are collected from recent literature and include heuristic, evolutionary and statistical methods. To provide a fair comparison, the algorithms are classified according to the approach used and examined against a large set of metrics. In particular, the effectiveness of each algorithm was evaluated for the optimization of a multi-processor platform, considering ini-

tial setup effort, rate of convergence, scalability, and quality of the resulting optimization. Our experiments are performed with statistical rigor, using a set of very diverse benchmark applications (a video converter, a parallel compression algorithm, and a fast Fourier transformation algorithm) to take a large spectrum of realistic workloads into account. Our results provide insights on the effort required to apply each algorithm to a target design space, the number of simulations it requires, its accuracy, and its precision. These insights are used to draw guidelines for the choice of DSE algorithms according to the type and size of design space to be optimized.

## 4.1 Introduction

The continuous increase of transistor density on a single die is leading towards the production of more and more complex systems on a single chip, with an increasing number of integrated components and processing units. This trend brought to the introduction of the System-On-Chip (SoC), that integrates on a single medium all the components of a full system. The design and development of such systems [123] raises challenges [66] due to the large design space, and tight constraints [99].

Parametrized embedded System-on-Chip (SoC) architectures must be optimally tuned, i.e., their configuration parameters must be appropriately chosen, to find the best trade-off in terms of the selected figures of merit (e.g., energy, area, and delay) for a given class of applications. This tuning process is called *Design Space Exploration (DSE)* [130]. Using DSE, a designer can find the optimal *configuration* for a given system.

In general, this optimization problem involves the minimization (or maximization) of *multiple objectives*, making the definition of optimality not unique. The quality of a system configuration according to the various objectives is usually expressed using a set of *metrics* or *objective functions*. Solving *multi-objective* optimization problems consists of finding the points of the *Pareto curve* [56], i.e., all the points which are better than all the others for at least one metric or objective function.

However, a Pareto curve for a specific platform is available only when all the points in the design space have been evaluated and characterized in terms of objective functions. This full search approach is often unfeasible due to the high cardinality of the design space, and to the high cost associated with the evaluation of the objective functions (e.g., long simulation times). A viable, but less expressive, solution would consist in the use of scalarizing functions [104]), which transform the problem into a single objective search.

Currently, Multi-Processor SoCs (MPSoCs) platforms are optimized using either by designer

experience or by applying several different algorithms. Examples are classical heuristic algorithms (such as tabu search, simulated annealing, etc.) [113], or pruning techniques that try to reduce the size of the design space [105]. These techniques rely on simulation (or estimation) for the evaluation of the system-level metrics corresponding to a newly found configuration.

Depending on the parameters of the design space (such as size and time needed for one simulation), algorithms show different performance and accuracy of results. This work identifies and compares 15 algorithms among several of the most recent approaches proposed in literature, for automated DSE with multiple performance metrics. These methods differ from theoretical background, applicability conditions, and performance. Through a rigorous analysis, we identify the advantages and drawbacks of each method, obtaining guidelines for their use with different applications.

The main contributions of this work are: (a) a comprehensive overview, as we take into account more and more diverse multi-objective optimization methodologies than previous surveys in the field, (b) a quantitative analysis, as our results provide numbers and metrics allowing a clear comparison of multi-objective optimization methodologies on the common ground of DSE for MPSoCs, and (c) insights, as we give recommendations for the choice of the most appropriate algorithm for a target design space.

The paper is structured as follows: Section 4.2 briefly reviews other comparative works in the field of multi-objective optimization and DSE; Section 4.3 describes the landscape of multi-objective optimization and DSE algorithms and proposes their partitioning into four classes; Section 4.4 summarizes the theory behind each one of the 15 algorithms included in our experimental comparison; Section 4.5 presents the experimental setup and the framework we used to compare the algorithms, while results are presented in Section 4.6; Section 4.7 contains the discussion of the results and the recommendations of the authors; finally, Section 4.8 concludes the paper.

## 4.2 Related Work

Several other works reviewing and comparing state-of-the-art algorithms for multi-objective optimization exist in literature [30, 181, 166], with most of them providing algorithm descriptions in the form of a survey. In general, these surveys do not attempt quantitative comparisons and they do not provide any novel experimental evaluation.

Fonseca and Fleming [48] claim that evolutionary algorithms (EA), since their appearance, quickly became one of the most popular ways to solve multi-objective problems. The paper

provides theoretical insights on how to assign fitness values [1] for multi-objective problems, but doesn't provide guidelines for the choice of EAs to be used on specific problems.

Coello [31] identifies three families of evolutionary approaches for multi-objective optimization based on the way the algorithms compute the fitness of individuals/solutions: algorithms using aggregating functions, algorithms using non-aggregating but non Pareto-based approaches, and Pareto-based approaches, e.g., the NSGA algorithm included in our experimental pool. Some experiments are reported, but the results are not systematically compared.

Marler and Arora [98] broadens the discussion on multi-objective optimization including approaches other than evolutionary algorithms. Multi-objective optimization methodologies are divided into: (a) algorithms with *a priori* specification of preferences (e.g., scalarizing and utility functions), (b) algorithms with *a posteriori* specification of preferences, returning a set of Pareto points, (c) algorithms with no specification of preferences, typically simplifications of approaches from (a) with constant parameters, and (d) genetic algorithms (GA), e.g., the IMMOGLS algorithm also described in this work. The cost of programming and the computational complexity of the algorithms are compared only in a qualitative way.

In terms of experimental groundwork, the closest works to this paper were published in [182], [180] and [181]. Zitzler *et al.* compare seven evolutionary algorithms, including SPEA and NSGA, detailed later in this paper, using one realistic (the synthesis of a multiprocessor) and two artificial problems. As our results confirm, they discover that SPEA outperforms NSGA. The main flaw of these publications is that their scope is limited to evolutionary algorithms, while we consider algorithms exploiting very different methodologies.

Finally, it is worth mentioning Okabe *et al.* [109], which summarizes several metrics for the evaluation of solutions to multi-objective optimization problems. Okabe *et al.* [109] suggests that no single metric is sufficient to judge the quality of the Pareto set returned by a multi-objective optimization algorithm. Similar claims, together with innovative visualization tools for the Pareto sets discovered by multi-objective evolutionary algorithms, can be found in [161]. In our work, we use four different metrics to evaluate the accuracy, distribution, and cardinality of the Pareto sets found by the optimization algorithms. Moreover, we compare the performance of 15 algorithms as the number of evaluations needed to converge to a Pareto set.

Table 4.1 Comparison of experimental settings in DSE literature.

| Ref. | Class | Approach | Optim. Metrics | | | Adjustable Parameters | | |
|---|---|---|---|---|---|---|---|---|
| | | | Delay | Power | Area | Proc. Units | Freq./Volt. | Cache Ass./Size |
| [111] | 1 | Particle Swarm Optimization | √ | √ | | | | √ |
| [94] | 1 | MO Pseudo Boolean | | √ | √ | √ | | |
| [56] | 1 | Parameter Dependency Model | √ | √ | | | √ | √ |
| [113] | 2 | Genetic Algorithms | √ | √ | | | | √ |
| [49] | 1 | Sensitivity-Based | √ | √ | | | | √ |
| [152] | 3 | Design of Experiments | √ | | √ | √ | | √ |
| [96] | 3 | Correlation-Based | √ | √ | | √ | | √ |
| [112] | 3 | Response Surface-Based | √ | √ | | √ | | √ |
| [15] | 4 | Decision Theoretic | √ | √ | | √ | √ | √ |

Table 4.2 Comparison of design space size in DSE literature.

| Ref. | Class | Design Space Size |
|---|---|---|
| [111] | 1 | 196608 |
| [56] | 1 | $> 10^{14}$ |
| [49] | 1 | 9216 |
| [113] | 2 | $5.97 \cdot 10^{12}$ |
| [152] | 3 | 16384 |
| [96] | 3 | $2^{17}$ |
| [112] | 3 | $2^{17}$ |
| [15] | 4 | 8640 |

### 4.3 Proposed Taxonomy

The challenge of automated DSE can be divided in two sub-problems: (a) the identification of candidate solutions (i.e., valid system configurations), (b) the evaluation of metrics of interest for such solutions, and the selection of the optimal configurations.

Considering the context of embedded systems design, we identify four main classes of techniques proposed in literature for the problem (a):

— **Class 1: Heuristics and pseudo-random optimization approaches,** that try to reduce the design space and focus the exploration on regions of interest [56, 49]. These techniques normally rely on full search or pseudo-random algorithms to explore the selected regions. Metaheuristics such as simulated annealing (SA) and multi-agent optimization belong to this class.

— **Class 2: Evolutionary algorithms.** These techniques are the most common and widely used, they rely on random changes of a starting set of configurations to iteratively improve the system under analysis. In this category we find genetic algorithms [113].

— **Class 3: Statistical approaches without domain knowledge.** These techniques extract a *metamodel* from the design space and use it to predict which new configurations to consider [152, 94, 112, 96].

— **Class 4: Statistical approaches with domain knowledge.** These techniques use pre-defined rules associated to the design space to find the most promising solutions [15].

Problem (b), i.e., the evaluation of candidate solutions, is usually addressed *via* two mechanisms (or a combination): detailed simulation [15] or estimation using predictive models [76].

**Class 1** features a vast range of algorithms that are currently used in various domains, including: (a) multi-agent optimization such as particle swarm optimization (PSO) [111], (b) simulated annealing, and (c) other operations research algorithms (such as tabu search). In [56], the design space is divided into partitions, and exhaustive search is performed inside each partition. Then the Pareto-optimal configurations of each partition are combined to determine the global curve. A different technique, proposed by Fornaciari *et al.* [49], aims at reducing the number of configurations from the product of the number of parameters to their sum. Although complexity is highly reduced, the exploration results remain in many cases sub-optimal.

**Class 2** techniques are probably the most widely used. Genetic algorithms and exact methods are combined in [94]: the design space exploration problem is formalized as a multi-objective

1. The metric used by evolutionary algorithms to evaluate the goodness of a solution and its probability of being recombined.

0-1 Integer Linear Programming (ILP) problem. A pseudo-boolean solver is used to force the genetic algorithm to stay in the feasible search space. These algorithms require a relatively small effort to describe design space, and they do not require specific knowledge associated to the domain or the metrics used for the exploration. However, they do not guarantee optimality or convergence within certain accuracy or precision bounds, even though they generally perform well when running a sufficient number of evaluations.

**Class 3** methods rely on statistical analysis. A group of techniques, referred to as *Design of Experiments* (DoE) [152, 112, 96] is often used to characterize the impact of the parameters on the system. This means estimating the portion of the variance of the objective functions associated to the variation of each parameter. Once sensitivity analysis is performed, heuristics or metamodels are used to modify the parameters and determine the optimal system configuration. Palermo *et al.* [112] use DoE to generate an initial set of experiments, creating a coarse view of the target design space. Response surface modeling (RSM) is then used to refine the exploration; this process is iterated to cover the design space. The solid statistical foundation of these algorithms allows to extract the maximum amount of information from the initial training set, and the generated metamodels can be used to find new candidate configurations or quickly evaluate potential solutions.

The algorithms in **Class 4** use expert knowledge to set up a probabilistic framework to determine the best candidate solutions. Beltrame *et al.* [15] introduce the use of decision theory to exploit this expert knowledge. The idea is to move the design space exploration complexity from simulation to probabilistic analysis of parameter transformations. Exploration is modeled as a Markov decision process (MDP) [143, 79], and the solution to such MDP corresponds to the sequence of parameter transformations to be applied to the platform to maximize (or minimize) a desired value function. This approach requires to simulate the system only in particular cases of uncertainty, massively reducing the simulation time needed to perform the exploration of a system, while maintaining the near-optimality of the results.

In this work we collected algorithms from all the four classes and we compared their strengths and weaknesses using a realistic benchmark scenario: a symmetric multi-processor platform. The results provide guidelines in the use of multi-objective optimization algorithms for DSE.

To allow a fair and clear comparison with previous DSE works, Table 4.1 reports the optimization metrics and parameters of previous publications, Table 4.2 contains information on the size of the explored domain spaces, while Table 4.3 briefly lists the software applications originally used as benchmarks.

Table 4.3 Comparison of benchmark applications in DSE literature.

| Ref. | Class | Benchmark Applications |
|---|---|---|
| [111] | 1 | **FIR** (finite impulse filter), **Gamma** and **DCT** (numeric algorithms), **Gauss** and **Quarcube** (equations solvers). |
| [94] | 1 | **ALC** (adaptive light control), a large automotive design problem. |
| [56] | 1 | **"JPEG"**, a *jpeg* compression algorithm using the on-chip DCT CODEC core to perform the forward DCT transform. |
| [49] | 1 | **MESA**, a 3D graphics library close to OpenGL, **GSM06.10**, European standard encoder and decoder. |
| [113] | 2 | **Motorola PowerStone**, a collection of embedded and portable applications. |
| [152] | 3 | A set of **EEMBC** (Embedded Microprocessor Benchmark Consortium) benchmark applications. |
| [96] | 3 | A set of benchmarks derived from the Stanford Parallel Applications for Shared Memory (**SPLASH**) suite. |
| [112] | 3 | **MPEG2** decoder application. |
| [15] | 4 | **ffmpeg**, **pigz** and **fft6** (see Table 4.9 for additional details). |

## 4.4 Multi-Objective Algorithms For Design Space Exploration

We include fifteen multi-objective algorithms for DSE in our experimental comparison.

### 4.4.1 Class 1

#### Adaptive Windows Pareto Random Search (APRS)

The APRS algorithm is one of the two novel algorithms implemented in the Multicube Explorer framework [174]. The algorithm starts with an initial set of candidate Pareto points and then attempts to improve the Pareto set by picking randomly among the points inside windows centered on the current points. The size of the windows is reduced over time (at each iteration) and proportionally with the quality of the point associated to them.

#### Multi-Objective Multiple Start Local Search (MOMSLS)

Local search is a simple heuristic that iteratively refines an initial random solution by looking for improvements in a neighbourhood of the current solution [143]. MOMSLS [78] simply consists of a local search procedure using multiple solution points in its initial step. At each step the algorithm looks for better solutions in the $N$ neighbourhoods of all the current solutions.

#### Multi-Objective Particle Swarm Optimization (MOPSO)

Particle Swarm Optimization (PSO) [83] is a heuristic search methodology that finds its biological inspiration in the behaviour of flocks of birds. At each search iteration, the particles in the swarm move towards an objective using a velocity vector that is the linear combination of three components:

— the previous velocity vector (weighted by inertia, $W$)

— the direction towards the best (i.e., closest to the objective) position ever reached by the swarm (weighted by a social learning factor, $C_1$)

— the direction towards the best position ever reached by the specific particle (weighted by a cognitive learning factor, $C_2$)

In multi-objective particle swarm optimization (MOPSO) [111], PSO is applied to a multi-objective domain by using $N$ swarms, each of which having as an objective the product of the multiple objectives combined with randomly chosen exponents.

In MOPSO, inertia and social learning factors are unused: in order to avoid local minima, at each iteration, particles are forced to move with random velocity (a random walk) with a fixed probability $p$.

### Multi-Objective Simulated Annealing (MOSA)

Simulated annealing (SA) is a local search technique that uses a special technique to avoid local minima: solutions that are worse than the current are rejected with a given probability $p$, computed using a Boltzmann distribution (parametrized by a coefficient $T$, called temperature).

[165] propose two ways of applying simulated annealing to multi-objective problems:

1. probability scalarization, the computation of rejection probability for each performance metric and their aggregation
2. criterion scalarization, the projection of the performance metrics into a single metric, and the use of such metric to compute the rejection probability of the new solution.

The latter approach is the one implemented in MOSA [165], and tested in this paper using the parameters in Table 4.5, while approach (1) is the one used by the other SA-based algorithms presented below.

### Pareto Simulated Annealing (PSA)

PSA [34] proposes two different criteria for the application of simulated annealing to multi-objective problems:

1. rule C says the rejection probability $p$ is proportional to the largest value among the differences between the performance metrics of the current and the new solution;
2. rule SL states that $p$ is a weighted linear combination of the differences between the performance metrics of the current and the new solution.

Table 4.4 The most relevant parameters used by the MOPSO algorithm.

| **MOPSO** [111] - Class 1 | | |
|---|---|---|
| **Param.** | **Description** | **Val.** |
| $W$ | inertia weight | *unused* |
| $C_1$ | social learning factor | *unused* |
| $C_2$ | cognitive learning factor | 1 |
| $p$ | probability of taking a *random walk* | $0.9, 0.5, 0.2$ |

Table 4.5 The most relevant parameters used by the MOSA, PSA and SMOSA algorithms.

| **MOSA** [165], **PSA** [34], **SMOSA** [150] - Class 1 | | |
|---|---|---|
| **Param.** | **Description** | **Val.** |
| $N_p$ | generating population size | 10 |
| $T_0$ | initial temperature used to parametrize the Boltzmann distribution | 2.50 |
| $T_f$ | final temperature used to parametrize the Boltzmann distribution | 0.1 |
| $\gamma$ | weights change coefficient | 0.1 |

Concerning rule SL, the weights used to multiply each metric are increased or reduced at each iteration, depending whether the most recently introduced solution brought a deterioration in that specific metric or not.

See Table 4.5 for the algorithm specific parameters used in our experiments.

#### Serafini's Multiple Objective Simulated Annealing (SMOSA)

Serafini [150] proposes several rules for the combination of multiple performance metrics in order to apply simulated annealing in the context of multi-objective optimization. Together with reviewing the C and SL rules described in [34], a new composite rule is introduced. This rule is the linear composition, with coefficients $\alpha$ and $(1 - \alpha)$, of two simpler rules:

1. rule P, saying that the rejection probability $p$ is proportional to the product of the differences between the performance metrics of the current and the new solution;
2. and rule W, saying that $p$ is proportional to the smallest value among the differences between the performance metrics of the current and the new solution.

### 4.4.2 Class 2

#### Multiple Objective Genetic Local Search (MOGLS)

MOGLS [72] is an algorithm combining two well known methodologies, genetic algorithms and local search. Algorithms combining multiple search methodologies are usually called hybrid approaches. Genetic algorithms start from a set of possible solutions, called population, and come up with new tentative solutions by combining couples of existing solutions $X$ picked with a probability $p$ given by a fitness function $f(X)$ [155]. At each iteration of

the MOGLS algorithm, (a) new solutions are generated using genetic operations and (b) a local search is performed in the neighbourhoods of these new solutions. The algorithm-specific parameters used in our experiments for MOGLS and the other genetic-local hybrid approaches, IMMOGLS and PMA, are reported in Table 4.6.

### Ishibuchi-Murata Multi-Objective Genetic Local Search (IMMOGLS)

IMMOGLS [73] is similar to MOGLS, being the combination of genetic algorithms and local search. In a multi-objective minimization problem, a solution is said to be non-dominated with respect to a set of solutions, if no other solution scores lower in all the metrics that we want to minimize, the solution is dominated otherwise. At each iteration, IMMOGLS performs both genetic operations and a local search with three characteristics:

1. the fitness function is a linear combination of the optimization metrics and the weights are chosen randomly at each iteration;
2. the local search is limited to a number of $k$ neighbours, where $k$ is random;
3. at each iterations, the current population is purged of any dominated solutions (elitist strategy).

### Non-Dominated Sorting Genetic Algorithm (NSGA)

NSGA [159] is a straightforward application of the genetic approach to multi-objective optimization. At each iteration, NSGA assigns to the Pareto points in the current population their fitness values on the basis of non-domination. All non-dominated solutions are assigned the same fitness value. See Table 4.7 for the algorithm specific parameters used in our experiments.

Table 4.6 The most relevant parameters used by the IMMOGLS, MOGLS and PMA algorithms.

| **IMMOGLS** [73], **MOGLS** [72], **PMA** [77] - Class 2 | | |
|---|---|---|
| **Param.** | **Description** | **Val.** |
| $N_p$ | population size | 20 |
| $N_i$ | number of iterations | 7 |
| $f$ | scalarizing function family | *linear* |

### Controlled Non-Dominated Sorting Genetic Algorithm (NSGAII)

Deb and Goel [36] present NSGAII, an evolution of NSGA with two main differences:

1. NSGAII is an elite-preserving algorithm, i.e., non-dominated solutions cannot be removed from the current population;
2. solutions are sorted based on non-domination to reduce computational complexity.

### Pareto Memetic Algorithm (PMA)

PMA [77] belongs to the family of hybrid genetic-local search algorithms, together with IMMOGLS and MOGLS. Its very own peculiarity resides in the way used to select the couple of current solutions that are combined with genetic operations: these two solution are not directly drawn from the current population. Instead, PMA samples with repetition a new set of solutions $T$ from the current population. Then, the best two solutions in $T$ are chosen for recombination.

### Strength Pareto Evolutionary Algorithm (SPEA)

SPEA [182] belongs to the broad family of heuristic search methods called evolutionary algorithms. Its peculiarities are:

1. all the non-dominated solutions are stored and preserved in a second external population;
2. the fitness of a solution in the current population is determined only from the solutions stored in the external non-dominated set [182];
3. a clustering step is applied to the non-dominated population in order to keep it small while preserving its characteristics.

Table 4.7 The most relevant parameters used by the NSGA, NSGAII and SPEA algorithms.

| **NSGA** [159], **NSGAII** [36], **SPEA** [182] - Class 2 | | |
|---|---|---|
| **Param.** | **Description** | **Val.** |
| $N_p$ | generating population size | 15 |
| $N_g$ | number of generations | 55 |
| $p$ | mutation probability | 0.2 |

### 4.4.3 Class 3

#### Response Surface Pareto Iterative Refinement (ReSPIR)

ReSPIR [112] is a DSE algorithm that uses statistical tools to create and keep an internal representation of the relations between the configuration parameters and the performance metrics in order to minimize the number of evaluations needed for successful optimization. These statistical tools are:

1. Design of experiments (DoE), a methodology allowing to maximize the information gained from a set of empirical trials;
2. response surface models (RSM), analytical representations of an objective trained with the available data. Different models can be used: linear regression, Shepard-based interpolation, artificial neural networks (ANN), etc.

The algorithm iteratively defines (using DoE) a set of experiments to be performed, trains the RSMs with the information collected and then produces an intermediate Pareto set.

### 4.4.4 Class 4

#### Markov Decision Process Optimization (MDP)

The approach proposed by Beltrame *et al.* [15] is based on a framework for sequential decision making called Markov decision process. Its components are states, actions, stochastic transitions from state to state, and rewards. The framework allows to find the correct action to perform in each state to collect the largest amount of rewards [143]. In [15], states are parameters configurations with their associated performance metrics, actions are parameters changes and rewards are improvements in the performance metrics. Stochastic transitions are initially believed to have uniform distribution and their estimates are refined over execution. It is worth noting that the algorithm requires built-in domain knowledge of the upper and lower bounds of each performance metric as a function of the tuning parameters. For this reason, MDP has a long set-up time and cannot be used without extensive domain knowledge regarding the platform used by the system under design. Table 4.8 reports the algorithm-specific parameters used in our experiments.

#### Multi-Objective Markov Decision Process (MOMDP)

MOMDP [16] is an improved version of MDP. The main difference with respect to MDP is that MOMDP uses a different exploration strategy. MDP considers different objectives using a parametric scalarizing function, and varies a parameter (called $\alpha$) which represents

Table 4.8 The most relevant parameters used by the MDP algorithm.

| **MDP** [15] - Class 4 | | |
|---|---|---|
| **Param.** | **Description** | **Val.** |
| $l$ | event horizon, maximum length of a decision path | 3 |
| $\epsilon$ | convergence margin, stopping criterion | $10^{-6}$ |
| $\lambda$ | accuracy factor, used to tune the discretization of the domain space | 0.3 |
| $\|A\|$ | the number of $\alpha$ values, i.e., number of modifications that can be applied to a parameter | 6 |

the weight(s) associated to each objective. By sweeping $\alpha$ values, it is possible to discover a Pareto curve for a given number of separate objectives.

MOMDP uses a different approach: it maximizes (or minimizes) one of the objectives to derive a starting point, and then builds the Pareto curve using a value function that selects a point that is close to the starting point, but improving it in at least one of the objectives. The process is repeated using the newly found point until a full Pareto front is discovered.

MOMDP also introduces a special action, called the leap of faith, that allows to avoid local minima by searching in the direction of high rewards, however unlikely. This action is performed when all actions fail to improve any of the metrics. The algorithm-specific parameters used in the experiments are the same as the ones in Table 4.8, with the exception of the event horizon, that was increased to 4.

## 4.5 Experimental Setup

We collected the implementation of the 15 multi-objective algorithms that we evaluated and compared from different sources. Most class 1 and class 2 algorithms algorithms are implemented in the Multiple Objective MetaHeuristics Library in C++ (MOMHLib++) by Jaszkiewicz and Dbrowski [78]. We also draw from the work of Zaccaria *et al.* [174], Multicube Explorer (M3Explorer), that implements standard and enhanced versions of several well-known multi-objective optimization algorithms. Multicube Explorer provides some of the DSE algorithms in classes 1 and 3. Concerning the algorithms in class 4, MDP and MOMDP, we used the original source code. All of these libraries and research projects are open-source and the authors provide open access to their implementations.

Our experiments aim at:

— determining the effort needed to apply each algorithm to a given design space, and

how the design space's characteristics drive the choice of the most effective exploration algorithm

— determining the number of evaluations required by each algorithm to obtain an approximate Pareto-set of given quality

— quantifying the properties of the Pareto-set found by each algorithm.

A qualitative comparison of the 15 algorithms is presented in Table 4.11. Each algorithm was applied to the same design space: a symmetric multi-processor platform running three different applications, shown in Figure 4.1. The platform consists of a collection of ARM9 cores with private caches, and a shared memory, interconnected by a simple system-bus model. Cache coherency is directory-based and using the MESI protocol. The ReSP [14] open-source simulation environment was used to perform the simulations, providing a set of configurable parameters, listed in Table 4.10. ReSP provides values for execution time and power consumption, which were used as the performance metrics for all optimization algorithms in our experiments.

The three applications used for testing are, more specifically, two large applications and a small benchmark, for which exhaustive search was possible, as listed in Table 4.9. *ffmpeg*, a video transcoder, was used to convert a small clip from MPEG-1 to MPEG-4, and *pigz*, a parallel compression algorithm, was used to compress a text file. The small benchmark consists of an implementation of Bailey's 6-step FFT algorithm (*fft6*). All applications are data-parallel and are targeted towards a homogeneous shared-memory multi-processor platform ($N$ processors accessing a common memory via bus). *ffmpeg* and *pigz* are implemented using pthreads, they create a set of working threads equal to the number of available processors and dispatch independent data to each thread. *fft6* uses OpenMP, with loop parallelization and static scheduling. These applications were specifically chosen in order to guarantee the maximum variability in their behaviour, in fact all applications use a different synchronization mechanism and require very different evaluation times.

The platform was explored using the parameters listed in Table 4.10 with a resulting design space of 8640 points [2], comparable with similar works (e.g., 6144 points in [152]). and the exhaustive exploration of any medium/large application would require an unfeasibly long simulation time (e.g., roughly two months for *ffmpeg*). Even the full exploration of the simple *fft6* benchmark required six days of uninterrupted simulation. To gather sufficient data for a statistical analysis, each of the 3 benchmark applications was optimized 10 times with each exploration algorithm ($N = 30$ executions for each algorithm).

2. It is worth noting that bus and memory latency are not realistic parameters, but they enlarge the design space to better test the proposed algorithm. The linear dependence with performance prevents any strong biasing of the results.

Table 4.9 The three benchmark applications chosen to represent a significant spectrum of workloads.

| **Application** | **Version** | **Source** | **Model** | **Synch.** | **Sim. Time** | **Description** |
|---|---|---|---|---|---|---|
| pigz | 2.1 | C | pthreads | condition | ∼2m | a parallel implementation of gzip |
| fft6 | 2.0 | C/Fortran77 | OpenMP | barrier | ∼30s | implementation of Bailey's 6-step fast Fourier transformation algorithm |
| ffmpeg | 49.0.2 | C | pthreads | semaphore | ∼30m | a fast video and audio converter |

Table 4.10 The platform design space simulated using ReSP.

| **Parameter Name** | **Domain** |
|---|---|
| # of PEs | {1,2,3,4,8} |
| PE Frequency | {100,200,250,300,400,500} MHz |
| L1 Cache Size | {1,2,4,8,16,32} KByte(s) |
| Bus Latency | {10,20,50,100} ns |
| Memory Latency | {10,20,50,100} ns |
| L1 Cache Policy | {LRU, LRR, RANDOM} |

According to Taghavi and Pimentel [161], the quality of the result of a multi-objective optimization algorithm is two-fold: 1) solutions should be as close as possible to the actual Pareto set, and 2) solutions should be as diverse as possible. Therefore, no single metric is sufficient in assessing the quality of the discovered Pareto set.

We use the three metrics presented in [43] to compare the relative quality of the approximate Pareto sets obtained by each algorithm:

— **ADRS** – The Average Distance from Reference Set is used to compare the approximated Pareto-sets with the best Pareto set found combining the results of all experiments. This approximates the distance of a considered set from the Pareto-optimal front, and should be minimized.

According to its definition in [174], the ADRS between an approximate Pareto set $\Lambda$ and a reference Pareto set $\Pi$ is computed as:

$$ADRS(\Pi, \Lambda) = \frac{1}{|\Pi|} \sum_{\boldsymbol{a} \in \Pi} \left( \min_{\boldsymbol{b} \in \Lambda} \{\delta(\boldsymbol{b}, \boldsymbol{a})\} \right) \tag{4.1}$$

where the $\delta$ function stands for:

$$\delta(\boldsymbol{b}, \boldsymbol{a}) = max_{j=1,..,m} \left\{ 0, \frac{\phi_j(\boldsymbol{a}) - \phi_j(\boldsymbol{b})}{\phi_j(\boldsymbol{b})} \right\} \tag{4.2}$$

Parameter $m$ is the number of objectives and $\phi_i(\boldsymbol{a})$ is the value of the $i$-th objective metric measured in point $\boldsymbol{a}$.

— **Non-uniformity** – We measure how solutions are distributed in the design space. Lower non-uniformity means a more evenly-distributed approximate Pareto-set that better estimates the optimal Pareto-set.

Given a normalized Pareto set $\bar{\Lambda}$, where $d_i$ is defined as the Euclidean distance between to consecutive points ($i = 1, .., |\bar{\Lambda}| - 1$), and $\hat{d}$ is the average values of all the $d$'s, non-uniformity [43] can be computed as:

$$\sum_{i=1}^{|\bar{\Lambda}|-1} \frac{|d_i - \hat{d}|}{\sqrt{m}(|\bar{\Lambda}| - 1)} \tag{4.3}$$

— **Concentration** – We measure the span of each Pareto-set with respect to the range of the objectives. The lower the concentration, the higher the spread of the Pareto-set and the better coverage of the range of objectives.

Given a normalized Pareto set $\bar{\Lambda}$, where $\phi_i^{min}$ is defined as $\min\{\phi_i(\boldsymbol{a})\ s.t.\ \boldsymbol{a} \in \bar{\Lambda}\}$ and

$\phi_i^{max}$ is defined as $\max\{\phi_i(\boldsymbol{a})\ s.t.\ \boldsymbol{a} \in \bar{\Lambda}\}$, concentration [43] can be computed as:

$$\prod_{i=1}^{m} \frac{1}{|\phi_i^{max} - \phi_i^{min}|} \tag{4.4}$$

## 4.6 Experimental Results

### 4.6.1 Dependence on Parameters and Initial Setup Effort

All the examined algorithms differ in the way they converge to an approximate Pareto front, and the quality of their results depends on a number of different parameters, making a fair evaluation difficult to implement. There is no common rule for the choice of each algorithm's parameters: these range from four to twelve, and they can be anything from integers to the choice of an interpolation function. The selection of the parameters that are optimal to a specific optimization problem requires either expertise or it can be calculated by meta-exploring (also known as parameter screening) the parameters on the target design space, i.e., running multiple explorations while changing the parameters to optimize the result. Either way, finding the optimal parameters usually requires trial and error. The Effort column of Table 4.11 qualitatively presents the tuning cost required by each algorithm.

Algorithms of classes 1 and 2 require few parameters (such as population size, mutation factors, initial temperature, etc.), and are generally robust to parameter choice. This means that small changes will not dramatically affect the outcomes of the exploration, although there is no guarantee that a given parameter choice will lead to optimal results. Their parameters have no direct link to any knowledge of the design space (e.g., initial temperature for MOSA and NSGAII), and can be determined only by experience or guesswork, rendering the best combination very difficult to obtain without screening and additional evaluations. In this work, we determined the best parameters via screening, which required running several thousand evaluations.

Algorithms like APRS do not require any special tuning, and rely on pre-determined heuristics, making their setup effort minimal. It is worth noting that both class 1 and 2 algorithms do not guarantee convergence to the optimal Pareto set and require that the user specify a maximum number of iterations in addition to any other stopping condition (e.g., when the results do not vary for more than two iterations).

Algorithms of class 3 demand a higher setup effort: the choice of a proper metamodel for a design space requires some expertise and an initial screening (and therefore additional evaluations) to properly determine which parameters are the most significant. Each metamodel

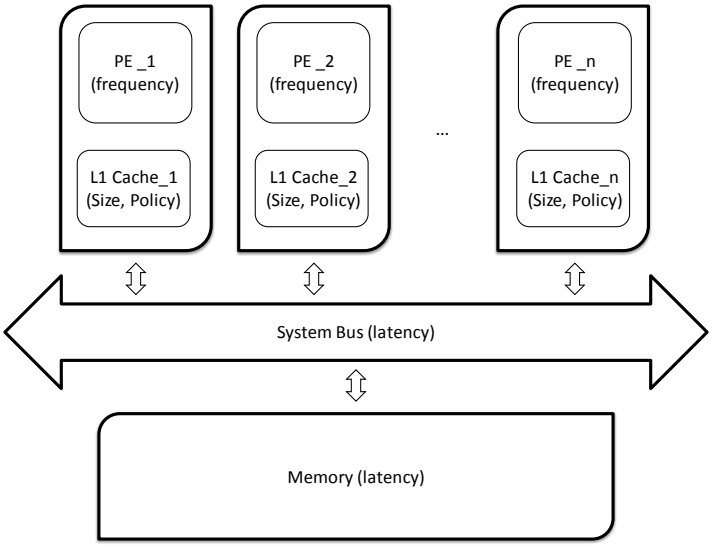



Figure 4.1 The simulated multi-core processor architecture and its parameters.

needs specific additional parameters that are loosely linked to the designer's expertise of the design space. For our experiments we relied on the results described in [112], and we chose the Central Composite Design using a neural network (NN) interpolator. However, the NN produced results with a very high variance, with ADRS ranging from 0% to 160%, making the use of this interpolator impractical. We found the Shepard interpolation much more effective, although it required to determine the value of a *power* parameter, which expresses how jagged is the response surface of the design space. A low value of this *power* parameter will produce a smooth interpolation, while a higher value could better follow a more jagged curve, but could also introduce overfitting. We effectively replicated the results of [112] on our design space using a power of 16, which was found by parameter screening ($\sim 10^3$ evaluations).

Finally, algorithms of class 4 require a bound to be associated to the effects of parameter variations on the configuration's metrics. These bounds are left to the designer's experience, or can be determined via statistical modelling. This means detailed analysis of each design space, and large setup effort. The main difference between MDP and MOMDP is that the former is fundamentally a single-objective optimization algorithm. To effectively discover a Pareto-set, MDP "sweeps" the design space according to a set of scalarizing values that express the desired trade-off between the different objective functions. To determine the size and values of these scalarizing values for our design space, hundreds of additional evaluations were necessary. MOMDP does not require this screening, and its only parameter (the accuracy $\lambda$) is in fact the average simulation error, and can be chosen without effort.

Table 4.11 A qualitative analysis of the chosen algorithms: setup effort, number of evaluations for 1% ADRS, number of Pareto points found, scalability.

| Acronym | Class | Effort | Evaluations | Pareto Points | Scalability |
|---|---|---|---|---|---|
| APRS [174] | 1 | ★ | ★★★★★ | ★★ | ★ |
| MOMSLS [78] | 1 | ★ | ★★★ | ★ | ★★★ |
| MOPSO [111] | 1 | ★★ | ★★★★ | ★★★ | ★★★★ |
| MOSA [165] | 1 | ★★★ | ★★★★ | ★★★ | ★★★★ |
| PSA [34] | 1 | ★★★ | ★★★ | ★ | ★★★★ |
| SMOSA [150] | 1 | ★★★ | ★ | ★ | ★★★★ |
| MOGLS [72] | 2 | ★★ | ★★★ | ★★★ | ★★★★ |
| IMMOGLS [73] | 2 | ★★ | ★★★ | ★★ | ★★★★ |
| NSGA [159] | 2 | ★★★ | ★★★ | ★ | ★★★★ |
| NSGAII [36] | 2 | ★★★ | ★★★★ | ★ | ★★★★ |
| PMA [77] | 2 | ★★ | ★★★ | ★★ | ★★★★ |
| SPEA [182] | 2 | ★★ | ★★★ | ★★★ | ★★★★ |
| ReSPIR [112] | 3 | ★★★★ | ★★ | ★★★★ | ★★★ |
| MDP [15] | 4 | ★★★★★ | ★ | ★ | ★★ |
| MOMDP [16] | 4 | ★★★★★ | ★ | ★★★ | ★★★ |

It is worth noting that designer experience can reduce or remove the need for parameter discovery activities for all the aforementioned algorithms.

### 4.6.2 Estimation of the Number of Evaluations

In our experiments, we have tuned each algorithm parameters to obtain the best results for design space described in this paper, and we **do not** include the evaluations needed for screening and initial parameter estimation. Concerning ADRS, since exhaustive search is not possible, we compare the Pareto set generated by each algorithm with the best Pareto set found compounding all evaluations performed by all algorithms, which covers a sizable portion of the entire design space (around 30%).

Figure 4.2 shows the percentage of the design space (i.e., the number of evaluations divided by the number of points in the design space) explored by each algorithm in order to reach an ADRS of approximately 1% on average. Note that it was not possible to have all the algorithms converge to the exact same quality result, and some algorithms show high variability. In fact, SMOSA and NSGA do not often converge to acceptable solutions. The final accuracy values obtained are shown in Figure 4.3. Please note that the histograms and the error bars in Figures 4.2 to 4.6 show average values and standard deviations, respectively, for

each algorithm over 30 experiments. No negative percentage was actually registered during the experiments

Figure 4.2 shows that the improvement can be worth the extra setup effort for class 3 and 4 algorithms: MDP, MOMDP and RESPIR have a factor 10 reduction in the number of evaluations, and a much tighter convergence (i.e., smaller variance of the results). Concerning class 2 algorithms, the performance is very similar, with IMMOGLS appearing to have the best combination of accuracy, number of evaluations and variance. APRS still provides excellent results given the zero-effort setup, although with at least twice as many evaluations when compared to class 2 algorithms.

### 4.6.3 Characteristics of the Resulting Approximate Pareto-set

Figure 4.4 shows the number of Pareto points found by each algorithm, normalized by the average found for each benchmark to allow for global comparison.

Most algorithms appear to not have any statistically significant difference, with the exception of RESPIR, which finds 25% more points than all the other algorithms on average, but with a slightly higher variance. Once again, SMOSA and NSGA display the poorest results. It is worth noting that some of the points found by RESPIR are Pareto-covered by the points found by the other algorithms: the number of points on the actual Pareto curve is smaller than what found by RESPIR.

Concerning non-uniformity and concentration, all algorithms behave similarly, covering the design space fully and without concentrating on specific areas. We could not report any statistically significant difference between the algorithms, with the only exception of SMOSA and NSGA, which show worse results coupled with high variance. Results are presented in Figure 4.5 and Figure 4.6.

### 4.6.4 Scalability

The scalability column of Table 4.11 refers to how the needed effort scales with the size of the design space: more scalable algorithms can be applied to more complex design spaces with lower effort. To test the effective scalability of each algorithm, we progressively increased the size of the design space, starting with three parameters and adding the remaining, three one by one. Similarly, we started with three values of each parameter and then increased their number progressively.

Most algorithms of classes 1 and 2 are very scalable: the number of parameters or the number of values per parameter do not affect the overall result, even though the number of additional

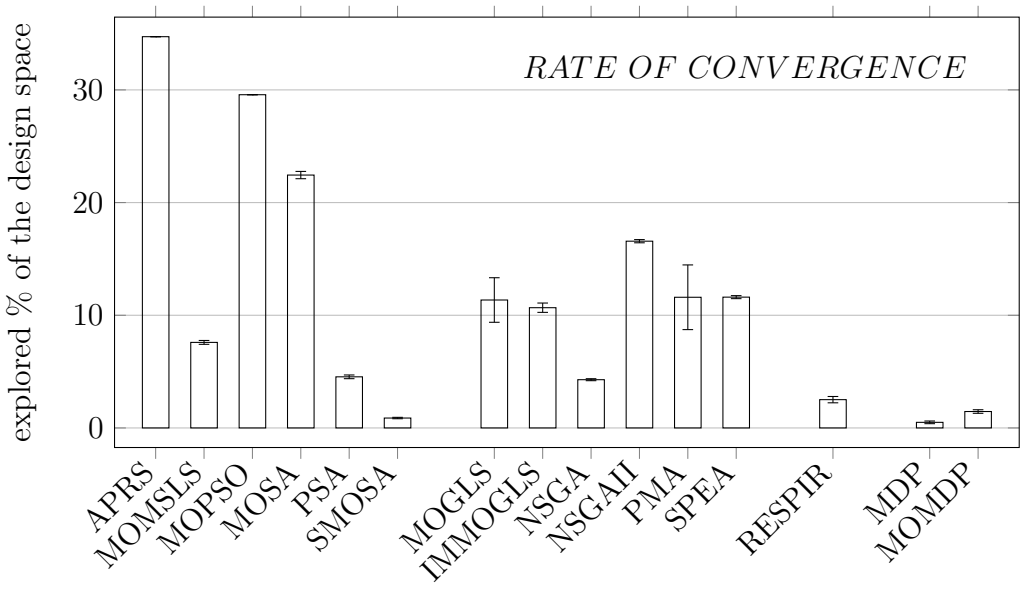



Figure 4.2 The percentage of points in the design space evaluated by each algorithm for similar levels of accuracy (around 1%).

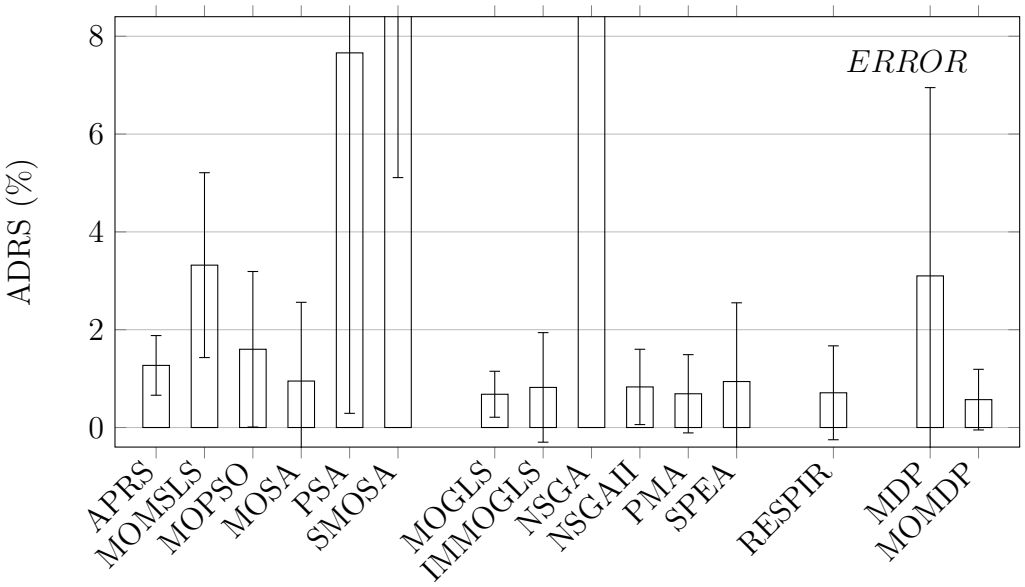



Figure 4.3 Accuracy (ADRS) reached by each algorithm at convergence.

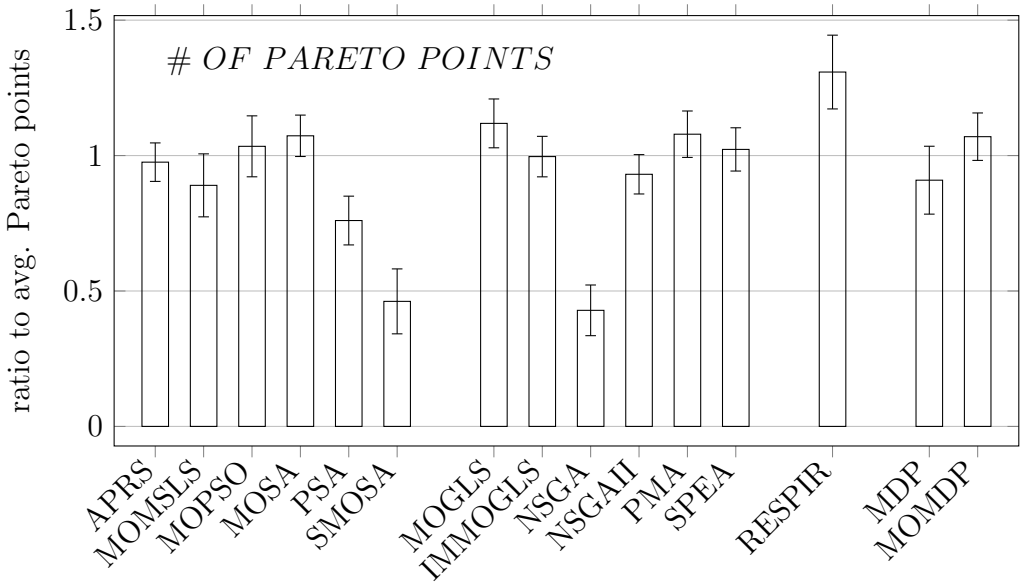



Figure 4.4 The average number of points in the approximate Pareto set found by each algorithm, normalized with the average for each benchmark.

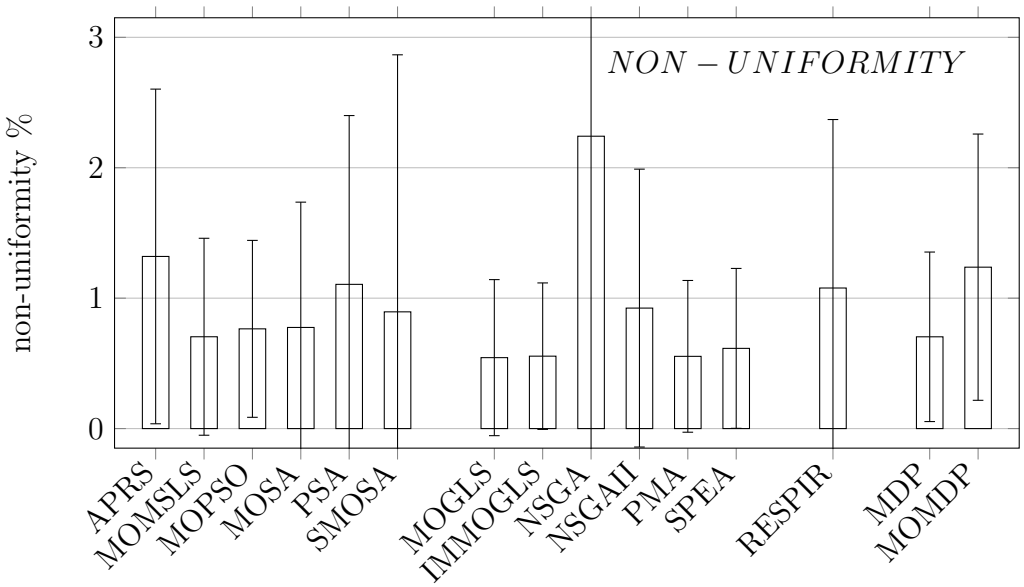



Figure 4.5 The non-uniformity of the distribution of the points found in the approximate Pareto set found by each algorithm.

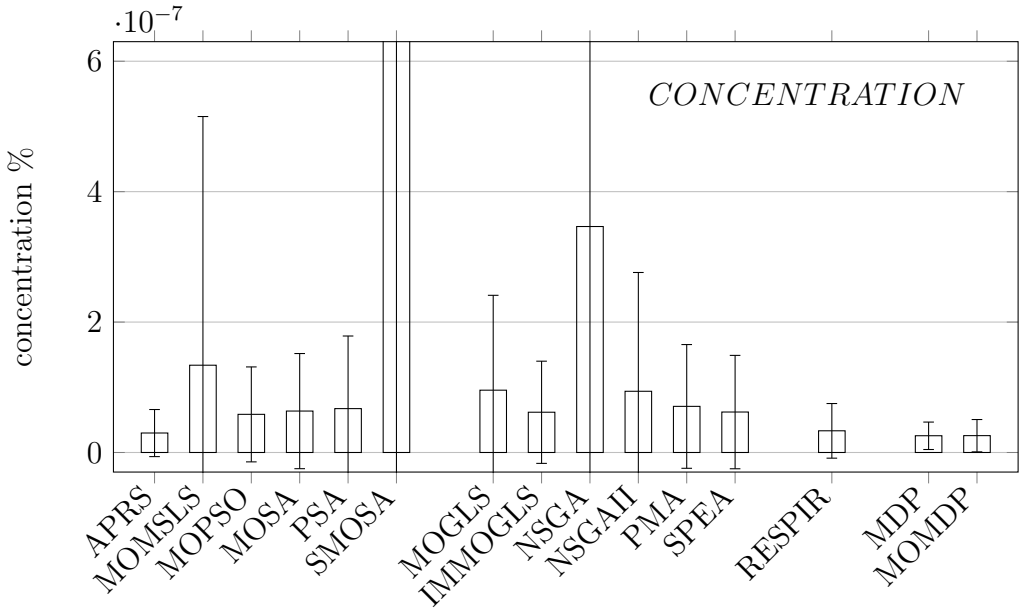



Figure 4.6 The concentration of the points found in the approximate Pareto set found by each algorithm.

evaluations needed grows proportionally with the number of parameters (i.e., remains around $\sim 10 - 15\%$ of the design space).

Although APRS require little setup effort, its applicability to large design spaces cannot be guaranteed. In fact, the already very high number of evaluations required increases exponentially with the design space size, practically limiting its use to small design spaces with fast evaluations.

RESPIR (class 3) scales well to design spaces with many parameters, but only if few values per parameter are present. This is due to one of the main limitations of central composite design: it can only consider three levels for each parameter, therefore reducing the accuracy of the method in presence of many parameter values, especially if they lead to non-linear behaviour. The number of evaluations for convergence remains $\sim 4\%$ of the design space.

Finally, class 4 algorithms scale orthogonally with respect to class 3: they scale well with the number of values per parameter, but not when the number of parameters increase. While adding a parameter require defining new bounds, adding new values comes without effort, and the number of evaluations needed increases less than linearly [15]. One drawback of MDP when compared to MOMDP is that it requires the estimation of an additional parameter ($\alpha$, see [15]) when increasing the size of the design space, which might require additional evaluations.

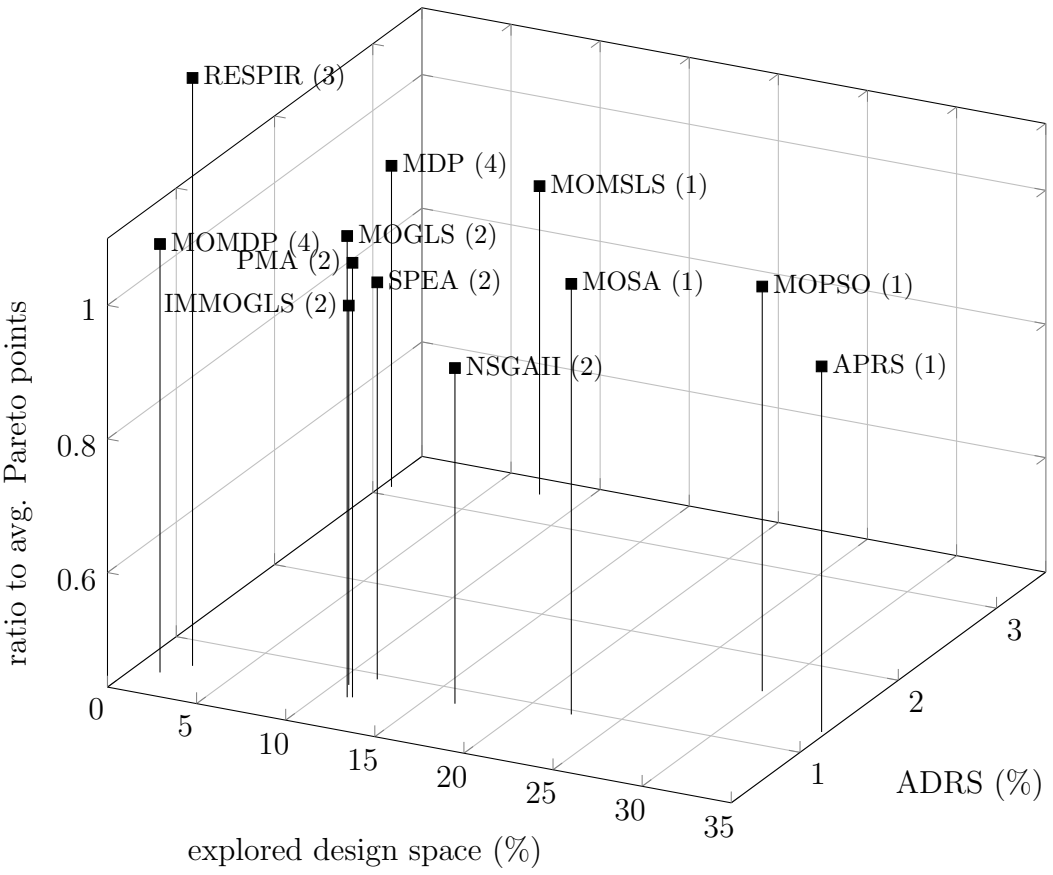



Figure 4.7 3D representation of the algorithms performance showing % of the design space explored in order to reach convergence, the number of Pareto points (w.r.t the average number) and the ADRS error metric (NSGA, PSA and SMOSA are omitted because of their large ADRS). The number after each algorithm name indicates the its class.

## 4.7 Discussion

The selection of the best algorithm for a particular application is difficult, and requires a trade-off between setup effort, scalability, expected number of simulations and expected accuracy. Given the experiments shown in Section 4.6, one can draw some general guidelines according to the cost of each evaluation (e.g., simulation time) and the size of the design space.

Higher evaluation costs make algorithms requiring a low number of simulations more appealing, even in case of a high upfront setup cost. On the contrary, if each evaluation has a very small cost, one might want to trade-off a higher number of evaluations with a no-effort setup. Similarly, large design spaces favour scalable algorithms, while smaller spaces do not justify the extra work to apply sophisticated algorithms.

In order to quantify the *actual* convergence time of an algorithm $i$ we have to take into consideration the design space size ($|S|$ points), the time required by each simulation/evaluation ($T_{sim}$ seconds), the percentage of the design space explored before reaching convergence ($\nu_i$) and the set-up time $SETUP_i$ of the algorithm:

$$\text{ACTUAL}_i = (\nu_i \times |S| \times T_{sim}) + \text{SETUP}_i \tag{4.5}$$

The values of $\nu_i$'s are reported in Figure 4.2. Regarding the $SETUP_i$ values, we translated the qualitative information in Table 4.11 into a 10'-to-30hrs range: we have observed during our experiments that algorithms having one "effort star" can be set-up in a few minutes while algorithms with five "effort stars" require more than a day of work.

Figure 4.8 presents four plots having the size of the design space on the x axis and the time required by each simulation on the y axis. In each plot, areas are labelled with the name of the more suitable algorithm according to Equation 4.5. Subplots (a) and (b) report the result for algorithms able to obtain ADRS of about 1%, whether domain knowledge is available (a), or not (b). Subplots (c) and (d) relax the ADRS requirement to about 5%.

Whether or not domain knowledge is available, Multi-Objective Multiple Start Local Search (MOMSLS) and Adaptive Windows Pareto Random Search (APRS) are the most appealing solutions for small to medium design spaces with non-expensive evaluations. As we explained in section 4.6, the APRS algorithm is a heuristic-based one and it requires very little setup effort, making it the ideal choice when simplicity is valued and there is no specific need for more efficient but also more complex approaches. MOMSLS is considerably faster but also more likely to produce a greater ADRS.

When domain knowledge is available, the Multi-Objective MDP algorithm (MOMDP) [16] clearly shows better performance, both in term of fast convergence to a very small ADRS and quality Pareto set, in large design spaces with high cost evaluations.

When one cannot obtain or exploit domain knowledge, the choice is split between the very high-quality Response Surface Pareto Iterative Refinement (RESPIR) algorithm and the Ishibuchi-Murata MO Genetic Local Search (IMMOGLS) algorithm. Both these algorithms are suited for very large design spaces.

However, it is worth noting that the IMMOGLS algorithm usually requires a larger number of evaluations, therefore is not recommended when dealing with high cost simulation. Pareto Simulated Annealing (PSA) is a valid alternative to IMMOGLS when a larger ADRS is acceptable.

What we can conclude from Figure 4.8 is that algorithms with small set-up times (i.e., the ones in classes 1 and 2) are especially suitable for simple problems with relatively small design spaces and/or short simulation times. On the other hand, complex algorithms in classes 3 and 4 always compensate for their longer configuration times when the exploration problem is difficult enough.

These recommendation are qualitative, but do take into account all the parameters shown in Section 4.6.

## 4.8 Conclusions

Concluding, this paper presented a classification and comparative analysis of fifteen of the best recent multi-objective design exploration algorithms. The algorithms were applied to the exploration of a multi-processor platform, and were compared for setup effort, number of evaluations, quality of the resulting approximate Pareto set, and scalability. The results give guidelines on the choice of the proper algorithm according to the properties of the design space to be addressed. In particular, we have determined the most promising algorithms when considering design space size and evaluation effort.

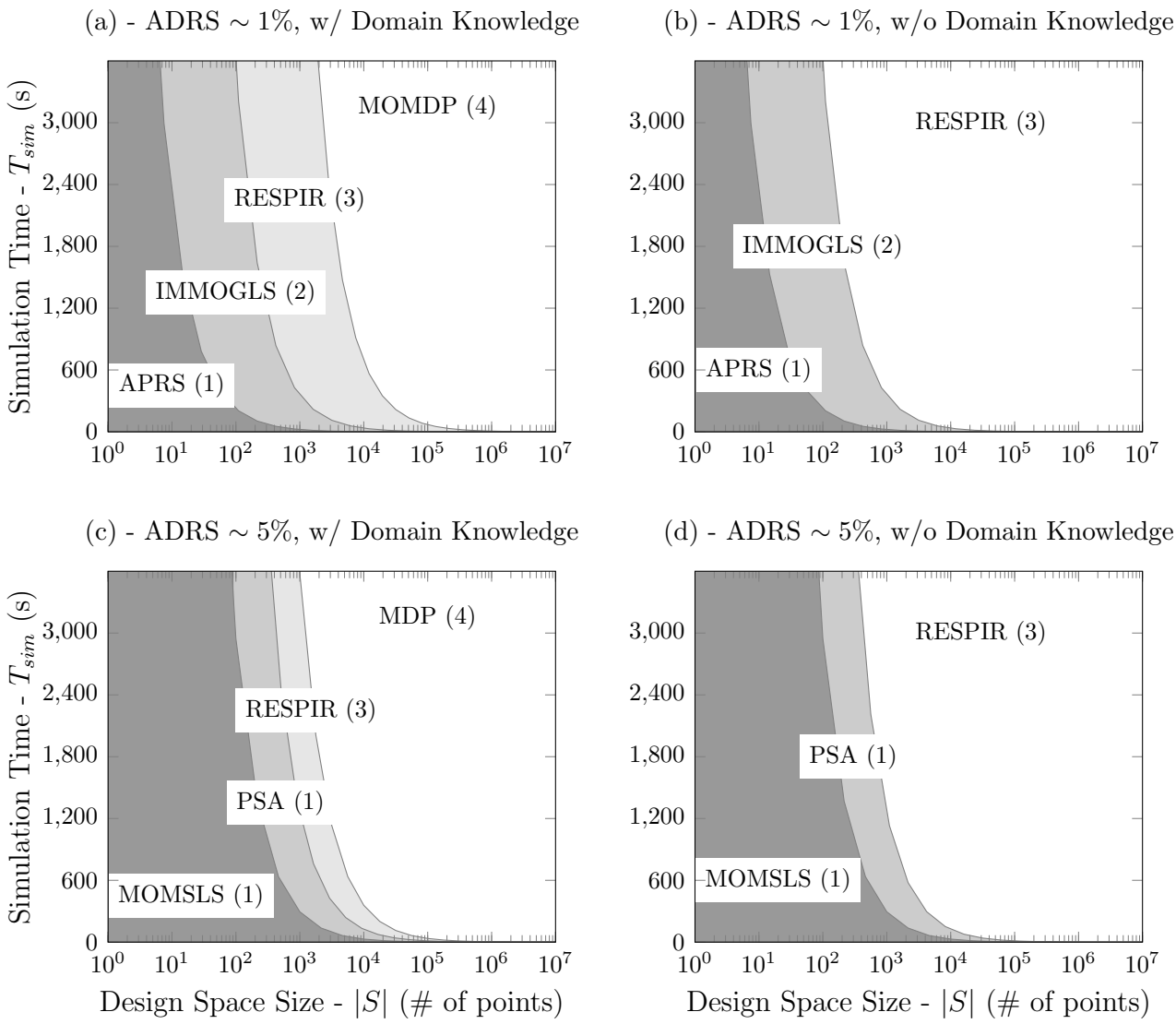



Figure 4.8 Recommended algorithms for different design space size, simulation time and desired ADRS, whether domain knowledge is available or not. The number after each algorithm name indicates its class.

# CHAPTER 5 ARTICLE 2 – BALANCING SYSTEM AVAILABILITY AND LIFETIME WITH DYNAMIC HIDDEN MARKOV MODELS

**Preface:** One of the problems that we chose to tackle early on with this research was the intrinsic inefficiency of the methodologies traditionally used by the aerospace industry to improve reliability—e.g., triple (or N) modular redundancy. Replicating a computing resource N times involves a N-00% overhead in cost, area, and power consumption. This expense can be misallocated if the mission profile only bears rare transient errors, while the entire system is still potentially compromised after `ceil`(N/2) permanent failures. In this chapter, we define and study the performance of a statistical model that combines knowledge of the orbit-dependant transient error rate and the expected lifetime of a resource to estimate whether a spacecraft's on-board computer is affected by a particular kind of error. Our results demonstrate that leveraging this knowledge (while using memory scrubbing for error detection), always outperforms traditional strategies that declare the death of a resource after a fixed number of errors. The lifetime improvement is potentially N-fold when compared to that of a N-modular scheme.

**Full Citation:** J. Panerati, S. Abdi and G. Beltrame, "Balancing system availability and lifetime with dynamic hidden Markov models," *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Leicester, 2014, pp. 240-247.

**DOI:** `https://doi.org/10.1109/AHS.2014.6880183`



**Abstract:** Electronic components in space applications are subject to high levels of ionizing and particle radiation. Their lifetime is reduced by the former (especially at high levels of utilization) and transient errors might be caused by the latter. Transient errors can be detected and corrected using memory scrubbing. However, this causes an overhead that reduces both the availability and the lifetime of the system. In this work, we present a mechanism based on *dynamic hidden Markov models* (D-HMMs) that balances availability and lifetime of a multi-resource system by estimating the occurrence of permanent faults amid transient faults, and by dynamically migrating the computation on excess resources when failure occurs. The dynamic nature of the model makes it adaptable to different mission profiles and fault rates. Results show that our model is able to lead systems to their desired

lifetime, while keeping availability within the 2% of its ideal value, and it outperforms static rule-based and traditional *hidden Markov models* (HMMs) approaches.

## 5.1 Introduction

At high altitude or in space, without the protection of the earth's magnetic field and atmosphere, integrated circuits are exposed to high level of radiation and heavy ion impacts that can disrupt the correct circuits' behaviour.

In this work we provide a mathematical framework to establish an adaptive system capable of managing both device aging (as accelerated by ionizing radiation) and *single event upsets* (SEUs), or soft errors, usually caused by the transit of a single high-energy particle through the circuit.

Detection of transient errors and protection against SEUs can be obtained in several ways [128], but there are no clear guidelines on how to identify permanent faults. One way to do it is to retry the same computation multiple times, and after a certain number of errors in a given interval of time, declare the component as permanently faulty [60].

This sort of on-line testing requires additional resources and inherently reduces the availability of the system. Moreover, the repetition of computation increases the strain on the electronics, reducing their lifetime. In this work we propose a framework that provides a system with the ability to decide, in case a fault is detected, if it is worthwhile to perform additional testing or if the component should be classified as permanently damaged.

The framework we introduce is a *failure detection mechanism* based on dynamic hidden Markov models . The main aspects of novelty of this work are:

— the extension of the static hidden Markov model framework with a dynamic transition model. This allows for the correct application of HMMs to the modeling of more general, *non-memoryless* failure processes;

— the D-HMM integration with a compact statistical modeling of transient and permanent faults occurring in electronic components exposed to ionizing and particle radiation;

— the definition of the lifetime-availability trade-off faced by failure detection mechanisms;

— the comparison, through simulation performed with real-life parameters, of the proposed D-HMM approach against traditional HMMs and rule-based systems.

The rest of the paper is structured as follows: we review relevant work in the field of fault detection in Section 5.2; Section 5.3 presents the theoretical background needed to understand our methodology, which is exposed, along two simpler alternatives, in Section 5.4;finally, the

setup used to evaluate the approach, the results discussion, and the conclusions are presented in Section 5.5, Section 5.6, and Section 5.7, respectively.

## 5.2 Related Work

This section reviews some important research work in the field of fault detection, automated classification and recovery. We also cover work dealing with the modeling of fault occurrences and failure times.

The need for computing systems tolerant to both software and hardware faults is not a novel one: reliability engineering provides us with a plethora of tools, mainly statistical ones, for the definition and analysis of such systems.

Creating fault-tolerant systems for space applications, however, is particularly challenging because of the several different types of faults that can arise when computers operate outside the atmosphere. Smolens *et al.* [156] called the attention on the fact that "aggressive CMOS scaling" results in an accelerated wear out of transistors and wires, and it inevitably leads to "shorter and less predictable lifetimes for microprocessors".

On the other hand, Karnik and Hazucha [80] studied how radiation particles interact with silicon and how these interaction should influence the design of VLSI systems. Radiation, in fact, can induce SEUs, impact system reliability, and it poses "a major challenge for the design of memories and logic circuits [...] beyond 90nm".

Cassano *et al.* [25] observed that SRAM-FPGAs represent a flexible and powerful resource for the creation of adaptive systems. However, their application in the context of aerospace creates the need for methodologies to detect (and cope) with both permanent and transient faults. In [25], they presents "a software flow for the generation of hard macros for [...] the diagnosing of permanent faults due to radiation".

Analogous remarks regarding the use of SRAM-FPGAs in space applications were made by Jacobs *et al.* [75]. In [75], we can find "a reconfigurable fault tolerance (RFT) framework that enables system designers to dynamically adjust a system's level of redundancy and fault mitigation based on the varying radiation incurred at different orbital positions". What is most relevant to our work, however, is the introduction of an "upset rate modeling tool" used to capture time-varying radiation effects in a given orbit.

The methodology we propose here is motivated by all of these works. Our transient fault model is more naive than the one described in [75], even if our approach could be easily extended to use it; on the other hand, unlike [75], our system is also capable of dealing with permanent faults. With respect to [25], we observe that our approach, not only detects both

permanent and transient faults, but it does that while maximizing lifetime and availability.

## 5.3 Theoretical Background

In the context of dependable computing, a *fault* is defined as the hypothesized cause of an *error*, which is itself defined as the deviation from the correct and desirable behaviour of a service [8]. With regard to persistence, faults can be classified in two categories:

— *transient faults*, whose negative effects on the system are assumed to be limited in time;

— *permanent faults*, which are assumed to last forever since the moment they appear, and may lead to the eventual *halt failure* of the system and the impossibility of providing a service if redundant resources or reconfiguration capabilities are not available.

In space, computing systems are exposed to high levels of radiation that pose a serious hazard to their proper functioning and survival. Ionizing and particle radiation, in fact, can be held responsible for a variety of undesirable outcomes. In particular, we distinguish:

— *single event upsets* (SEUs), these are transient faults that cause changes in the content of individual memory elements, i.e., *bit flips*;

— long-term damages, caused by the *total ionizing dose* (TID), that can have disruptive effects on current CMOS technologies and lead to performance degradation, permanent faults, and system failure [25].

*Data scrubbing* is an error detection and correction technique that consists in periodically re-reading and re-writing the content of a memory, using a "safe" copy known to be correct [54]. Exploiting data scrubbing, we are given the ability to detect faults, and, if they are not permanent, to correct them, at constant time intervals of duration $T$ (scrubbing period).

### 5.3.1 Transient Faults Modeling

We employ probability theory to model the occurrence of transient (SEUs) and permanent (i.e., system failures given by TID) faults in space computing systems. The impacts of high-energy particles that cause SEUs are known to be independent and they usually happen at a constant average rate, given by the orbit or mission phase of the system.

We define the probability of observing at least one SEU in a scrubbing period of size $T$, given a constant average rate of $SEU_r$, as $P_{SEU}$. The probability of observing a given number of events that are known to occur at a constant average rate $r$ is described by the Poisson distribution of parameter $r$, $Pois(r)$. Therefore, $P_{SEU}$ does not depend on the actual time

or history of the system and can be computed as:

$$\begin{aligned} P_{SEU} &= P(C_{SEU}(T) \geq 1 \mid C_{SEU}(T) \sim Pois(SEU_r \cdot T)) \\ &= 1 - P(C_{SEU}(T) = 0 \mid C_{SEU}(T) \sim Pois(SEU_r \cdot T)) \\ &= 1 - \text{pmf}_{Pois(SEU_r \cdot T)}(0) \\ &= 1 - e^{-SEU_r \cdot T} \end{aligned} \tag{5.1}$$

where $C_{SEU}(T)$ is defined as the number of SEUs observed in a period $T$, and pmf is the probability mass function. Figure 5.1 shows that this probability quickly increases with $SEU_r$, if $T$ is large.

### 5.3.2 Permanent Faults Modeling

To model permanent faults we start by defining the probability of a permanent fault occurring in a component by time $t$, i.e $failure \leq t$, given that the component was still functional at the end of the previous scrubbing period $t - T$, i.e $failure > t - T$, as $P_{failure}(t, T)$.

This probability can be computed, using the Kolmogorov definition, as:

$$\begin{aligned} P_{failure}(t, T) &= P(failure \leq t \mid failure > t - T) \\ &= \frac{P(failure \leq t \wedge failure > t - T)}{P(failure > t - T)} \\ &= \frac{CDF_{failure}(t) - CDF_{failure}(t - T)}{1 - CDF_{failure}(t - T)} \end{aligned} \tag{5.2}$$

where $CDF_{failure}$ is the cumulative density function of the random variable $failure$ describing the time at which the failure happens.

In literature, several probability distributions are used to model failure times [42]. One of the most frequently used, because of its simplicity and the convenient *memorylessness* property, is the *exponential* distribution.

If we model the failure time using an exponential distribution with mean equal to expected mean time to failure (MTTF) of the system, Equation 5.2 becomes:

$$\begin{aligned} P_{failure}(t, T) &= \frac{1 - e^{-t/MTTF} - (1 - e^{-(t-T)/MTTF})}{1 - (1 - e^{-(t-T)/MTTF})} \\ &= 1 - \frac{1}{e^{T/MTTF}} \end{aligned} \tag{5.3}$$

Because the exponential distribution is memoryless, this value does not depend on current

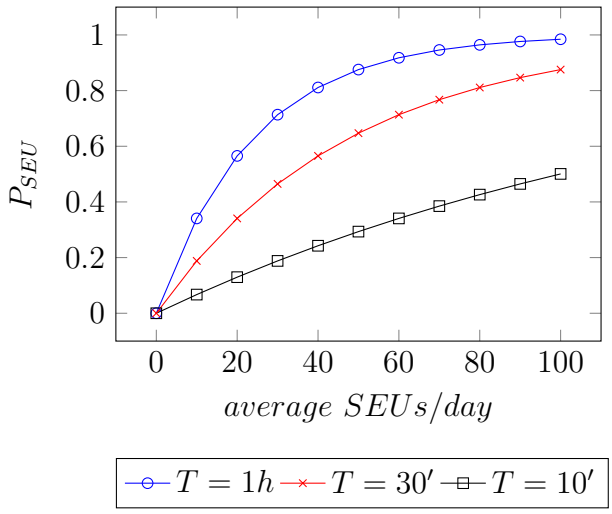



Figure 5.1 Probability of observing at least one SEU in scrubbing periods of different durations, as the average ratio of SEUs per day increases.

time $t$.

However, the exponential distribution representation is somewhat imprecise because it lacks the ability to capture the increasing failure probability due to accumulated wear in the component [42]. A common alternative used to overcome this limitation is a *log-normal* failure distribution:

$$P_{failure}(t,T) = \frac{\Phi\left(\frac{\ln t-\mu}{\sigma}\right) - \Phi\left(\frac{\ln(t-T)-\mu}{\sigma}\right)}{1 - \Phi\left(\frac{\ln(t-T)-\mu}{\sigma}\right)} \tag{5.4}$$

where $\Phi$ is the CDF of the normal distribution, and the log-normal parameters $\mu$ and $\sigma$ can be computed from the MTTF and the variance of the failure time as follows:

$$\mu = \ln\left(\frac{MTTF^2}{\sqrt{var_{MTTF} + MTTF^2}}\right)$$

$$\sigma = \sqrt{\ln\left(1 + \frac{var_{MTTF}}{MTTF^2}\right)}$$

We can observe that in Equation 5.4, $P_{failure}(t,T)$ is no longer independent of the actual time $t$.

Figure 5.2 summarizes the comparison of exponentially and log-normally distributed failure times with the same MTTF of 5 years. The rightmost chart, in particular, shows how a memoryless and a non-memoryless distribution function differently describe the occurrence

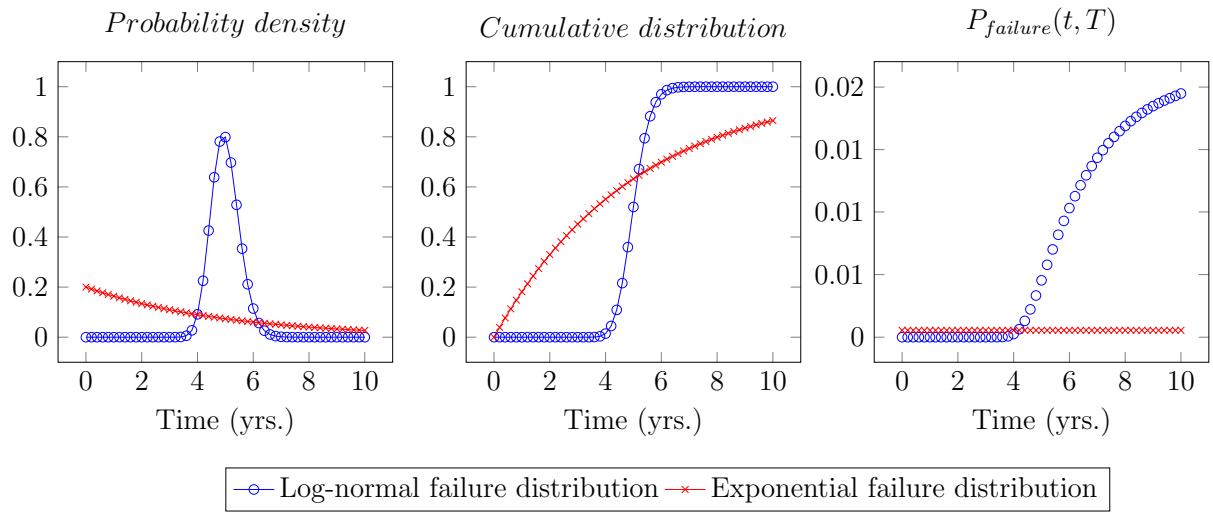



Figure 5.2 Probability density function, cumulative distribution function, and probability of failure in the last scrubbing period (see Equation 5.2, $T = 1$ day), for log-normal and exponential failure distributions with $MTTF = 5$ years.

of a recent failure.

### 5.3.3 Hidden Markov Models

A hidden Markov model is a mathematical framework capable of describing the evolution over time of a stochastic system that can only be indirectly observed through stochastic sensors. An HMM can be considered as a special case of a Dynamic Bayesian Network (DBN) [143].

In an HMM, time is discrete and, at each time time step $t$, the state of the system is fully described by a discrete probability distribution function over a single random variable $S_t$. This variable cannot be observed, but its initial probability distribution $P(S_0)$ is known. At each time step, one can observe *evidences* $O_t$ that solely depend on the current probability distribution over all the possible states $P(S_t)$. Moreover, the process is 1-step Markovian, meaning that $P(S_t)$ only depends on $P(S_{t-1})$. Figure 5.3 shows the structure of an HMM over 3 time steps.

In summary, HMMs can be formally described as:

— $P(S_0)$, an initial probability distribution over all the possible states;

— $T_{ij} = P(S_t = j \mid S_{t-1} = i)$, a transition model, governing the evolution of the system;

— $E_{ij} = P(O_t = j \mid S_t = i)$, a sensor model, that links the current state and possible observations.

**State Prediction and Estimation using HMMs**

HMMs can be used to predict the probability of the system being in a specific state at time $t$, given all the evidence up to time $t-1$, using the following equation (that can be recursively applied, having its base case in the initial probability distribution $P(S_0)$):

$$P_t(S = j \mid O_{1:t-1}) = \sum_i P_{t-1}(S = i \mid O_{1:t-1}) \cdot T_{ij} \tag{5.5}$$

Moreover, an HMM prediction can be refined after the observation of the current evidence by applying:

$$P_t(S = i \mid O_t = j,\ O_{1:t-1}) = \alpha\ P_t(S = i \mid O_{1:t-1}) \cdot E_{ij} \tag{5.6}$$

where $\alpha$ is a normalization parameter.

## 5.4 Proposed Approach

In our approach, we use a multi-resource system model, and three methodologies to balance its lifetime and availability: a simple rule-based approach, a standard HMM and the proposed dynamic hidden Markov model formulation.

### 5.4.1 System Model

For our analysis, we introduce a simple computing system model, shown in Figure 5.4, with the following assumptions:

— the system is composed by $N$ identical resources $R_i$'s;

— each resource $R_i$ is, alone, capable of providing the service required from the system;

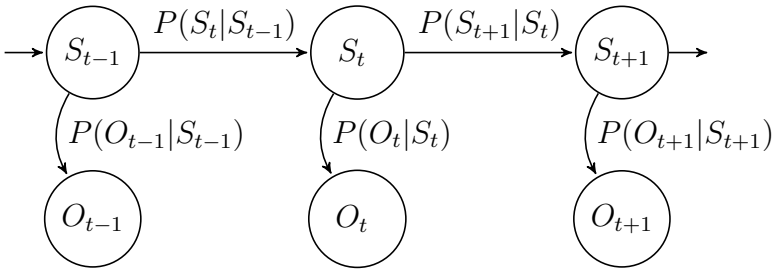



Figure 5.3 Structure of a hidden Markov model.

— the system only uses one resource $R_i$ at a time (to maximize system lifetime: in fact, resource wear depends on the resource being turned on and not on its level of utilization [64]);

— when a resource is active, it is subject to transient faults with constant average rate $SEU_R$;

— the failure time of each resource follows a log-normal probability distribution with mean equal to one N-th of the mean time to failure of the entire system ($MTTF$), starting from the time the resource is used for the first time. This means that the resource is not subject to wear as long as it is inactive;

— the active resource performs data scrubbing with a constant period $T$;

— whenever the scrubber detects the occurrence of and error, all the computation performed in the last period is discarded;

— whenever the scrubber detects the occurrence of and error, a *failure detection mechanism* is in charge of deciding whether to *rollback* the computation on the same resource $R_i$ or to declare the resource dead (in permanent fault mode) and *migrate* the computation on the following resource $R_{i+1}$;

— migration happens at a cost of a migration time $T_{migr.}$;

— the decision of declaring the death of a resource is irrevocable;

— after the last resource $R_N$ is declared dead, the system is itself declared dead.

We can think of this model as a homogeneous multi-core system or any other N-times modular redundant computing system. It is clear from these premises that a *failure detection mechanism* that hastily declares the failure of resources will negatively impact the total lifetime of the system. However, a mechanism lingering for too long on obvious decisions might reduce the actual availability of the system.

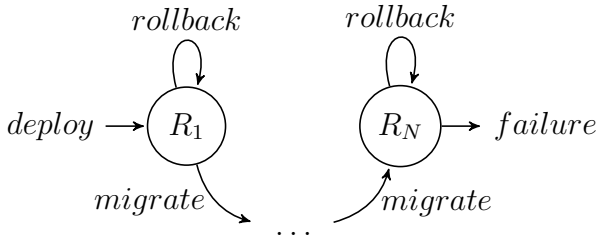



Figure 5.4 Structure of an N-resource system.

### 5.4.2 Rule-based Failure Detection

To decide whether a resource should be considered in a state of permanent fault, we can employ different failure detection mechanisms. For this purpose, several previous research works exploit rule-based systems of different complexity [60, 75].

The simplest rule-based approach, implemented in this work as a reference (Algorithm 1), is based on the assumption that a resource failure will cause consecutive faults. If the number of consecutive faults observed by the mechanism is larger than a predefined threshold, the resource is assumed to have failed, and the computation is migrated to another resource, if available. The threshold can be as little as 2, meaning that two consecutive faults are enough to assume a permanent failure, or much larger. A threshold of 32, coupled with a scrub period $T$ of 1 hour means that we are willing to wait over a day before declaring the status of permanent fault.

```
1  consecutive_faults = 0;
2  available = 1;
3  while available do
4  |   wait(scrub_time);
5  |   fault = observe_fault();
6  |   if fault then
7  |   |   consecutive_faults ++;
8  |   |   if consecutive_faults ≥ threshold then
9  |   |   |   available = 0 ;
10 |   else
11 |   |   consecutive_faults = 0;
12 |   end
13 end
```

**Algorithm 1:** Rule-based Algorithm

### 5.4.3 HMM-based Failure Detection

Algorithm 2 describes the behaviour of an HMM-based failure detection mechanism. In this case, the mechanism retains a *belief state*, i.e., a probability distribution over three possible states of the resource: (1) available, (2) experiencing a SEU or (3) failed.

At each scrubbing period, this belief state is updated using the transition model given in Table 5.2, and Equation 5.5. Then, after the result of the scrubbing operation, the belief state is filtered using the sensor model in Table 5.1, and Equation 5.6. Finally, Equation 5.5 is used again to predict the future state of the resource.

Table 5.1 HMM sensor model.

| | $P(O_t\|S_t)$ | |
|---|---|---|
| $S_t$ | Fault | No Fault |
| Available | 0 | 1 |
| SEU | 1 | 0 |
| Failure | 1 | 0 |

Table 5.2 HMM transition model.

| | $P(S_t\|S_{t-1})$ | | |
|---|---|---|---|
| $S_{t-1}$ | Available | SEU | Failure |
| Available | $1 - P(SEU \vee Failure)$ | $P_{SEU}$ | $1 - \frac{1}{e^{T/MTTF}}$ |
| SEU | $1 - P(SEU \vee Failure)$ | $P_{SEU}$ | $1 - \frac{1}{e^{T/MTTF}}$ |
| Failure | 0 | 0 | 1 |

The decision of performing a migration is taken if the probability that the system will be in permanent failure in the near future is greater than a predefined level of confidence.

Tables 5.2 and 5.1 show that we are assuming the sensor model to be perfect (only 0s and 1s in Table 5.1), the environment to be *static* (constant values in Table 5.2), and that permanent faults cannot be recovered (only 0s and a 1 in the last line of Table 5.2).

```
1  belief_state = init_belief();
2  available = 1;
3  while available do
4  |   wait(scrub_time);
5  |   fault = observe_fault();
6  |   if fault then
7  |   |   belief_state = update_belief(transition_model) ;
8  |   |   belief_state = filter_belief(sensor_model) ;
9  |   |   predict_state = predict_belief(transition_model) ;
10 |   |   if predict_state.failure ≥ threshold then
11 |   |   |   available = 0 ;
12 |   else
13 |   |   belief_state = update_belief(transition_model);
14 |   |   belief_state = filter_belief(sensor_model);
15 |   end
16 end
```

**Algorithm 2:** HMM-based Algorithm

### 5.4.4 Dynamic HMM-based Failure Detection

A major limitation of the HMM-based mechanism is that the transition model is constant over time. This means that the model could easily capture the behaviour of a resource with an exponentially distributed failure time, but it would fail to represent the log-normal distribution that we assumed in Subsection 5.4.1.

HMMs are meant to describe Markovian processes, i.e., processes that propagate one time step at a time. However, nothing prevents us from changing their static nature and make the transition model *dynamic*. We implemented the new transition model shown in Table 5.3: the first two elements of the last column, representing the probability of encountering a permanent failure in a single time step, are no longer constants, but rather a function of the global time variable. This allows us to exploit Equation 5.4 and properly model the log-normal failure time distribution.

It is worth noting that the algorithm implementing the modified dynamic HMM-based mechanism differs from Algorithm 2 only by an additional step used to re-evaluate the transition model. The sensor model is still assumed to be perfect, as presented in Table 5.1. As for the HMM-based mechanism, the decision of performing a migration is taken if the evidence shows that the system will be in permanent failure in the near future with a certain level of confidence.

Table 5.3 Dynamic HMM transition model.

| | $P(S_t \mid S_{t-1})$ | | |
|---|---|---|---|
| $S_{t-1}$ | Available | SEU | Failure |
| Available | $1 - P(SEU \vee Failure)$ | $P_{SEU}$ | $P_{failure}(t)$ |
| SEU | $1 - P(SEU \vee Failure)$ | $P_{SEU}$ | $P_{failure}(t)$ |
| Failure | 0 | 0 | 1 |

## 5.5 Experimental Setup

In order to perform the simulation and validation of our methodology we implemented the algorithms and the system model described in Section 5.4 using the MATLAB-compatible, GNU Octave programming language. Simulations where carried out on a quad-core Intel i7 desktop running at 3.2GHz with 32GB of RAM.

### 5.5.1 Parameters

The parameters that define the environment are those outside the control of the designer of the failure detection mechanism. These parameters regulate the occurrence of faults and the maximum lifetime of the system:

— $SEU_r$ - in our simulations, the average rate of single event upsets *per day* can take the value of 16.5 or 62. These are the number of daily SEUs expected in a Virtex-4 FX60 in a low-Earth orbit (LEO) and in a highly-elliptical orbit (HEO), respectively [75];

— $MTTF$ - we choose components so that the mean time to failure of the system is fixed to 5 years, each resource of the system has a MTTF equal to one N-th of 5, depending on the number of resources in the system;

— $var_{MTTF}$ - the variance of a resource lifetime is arbitrarily fixed to the 10% of the MTTF;

— $T_{migr.}$ - the migration time is assumed constant and equal to 10 minutes.

Furthermore, we perform our simulations screening multiple values for the parameters that can be chosen by the designer of the failure detection mechanism. These parameters affect the actual lifetime of the system and its availability:

— $N$ - the number of resources varies from 3 to 10;

— $T$ - we consider 3 different scrub periods of 10, 30, and 60 minutes each;

— finally, we consider 5 different thresholds for each failure detection mechanism: 2, 4, 8, 16, 32 for the rule-based method, and 0.5, 0.75, 0.88, 0.94, 0.97 for HMM and D-HMM.

### 5.5.2 Performance Metrics

As mentioned in Subsection 5.4.1, a good failure detection mechanism does not shorten the total system lifetime while maintaining an optimal system availability. Therefore, we consider lifetime and availability as the two metrics needed to assess the performance of our mechanisms.

The system lifetime $L$ is equal to the time $t$ when the mechanism declares the permanent failure of the last ($N$-th) resource in the system. This can be compared to the expected lifetime obtained by a *clairvoyant* mechanism given by equation:

$$\mathbb{E}_{cv}[L] = N \cdot MTTF + (N-1) \cdot T_{migr.} \tag{5.7}$$

An optimistic mechanism would record longer lifetimes, postponing the declaration of failure of computing resources as much as possible. This would happen at the price of a lower system

availability.

The total availability can be defined as the fraction of scrubbing periods in which no faults are detected:

$$A = 1 - \frac{|detected\ faults| \cdot T}{L} \tag{5.8}$$

However, this metric is biased towards small scrubbing periods $T$, and does not allow for a fair comparison of results obtained with varying SEU rates. Therefore, in this work, we use a normalized measure of availability defined as:

$$\widetilde{A} = \frac{A}{\mathbb{E}_{cv}[A]} \tag{5.9}$$

where the expected availability $\mathbb{E}_{cv}[A]$ is computed as:

$$\mathbb{E}_{cv}[A] = 1 - \frac{SEU_r \cdot N \cdot MTTF + (N-1) \cdot T_{migr.}}{N \cdot MTTF + (N-1) \cdot T_{migr.}} \tag{5.10}$$

## 5.6 Discussion

Figures 5.5 and 5.6 show the results obtained with $N = 10$ resources and a scrubbing period $T = 1$ hour, for a system operating in Low Earth Orbit (LEO, 16.5 SEUs per day) and a Highly Elliptical Orbit (HEO, 62 SEUs per day), respectively.

One can notice that the rule-based approach fails to achieve the desired lifetime of 5 years even at LEO, unless a very high threshold on the number of acceptable consecutive faults is used. Moreover, small thresholds result in short and highly variable lifetimes for both the rule-based and the HMM system. Regardless of the mechanism used, one can observe the presence of a compromise between availability and lifetime.

We also remark that the D-HMM approach performs better than HMM, especially in terms of lifetime. This can be explained by the fact that the D-HMM truly captures the accumulated wear of electronic devices. The very small deviations from the ideal normalized availability of 1.0 are justified by the observation that any failure detection mechanism can make an impact on this metric only when a failure actually happens, and its significance is limited by the number of resources $N$. We envision that much larger gains will be in reach in the many-cores era, when system will have tens or hundreds of resources. Figure 5.7 presents the lifetime-availability trade-offs obtained by the three failure detection mechanisms in the form of Pareto curves for LEO and HEO. The number of resources is again $N = 10$, and data scrubbing is performed with a period $T = 10$ minutes. To map the *ideal* situation in the origin of the axes $(0, 0)$ instead of $(+\inf, +\inf)$, we plot using the reciprocals of the

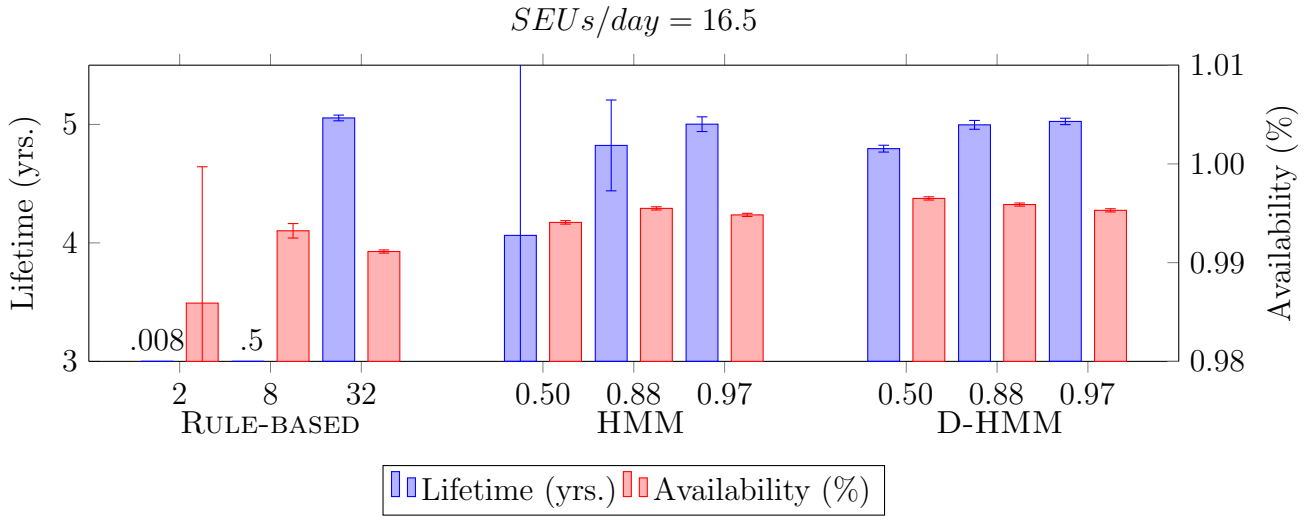



Figure 5.5 Comparison of system lifetimes and normalized availabilities (see Equation 5.9) for the rule-based, HMM-based, and dynamic HMM-based approaches with different thresholds, assuming $N = 10$ resources and a scrubbing period $T = 1$h, when SEUs/day = 16.5.

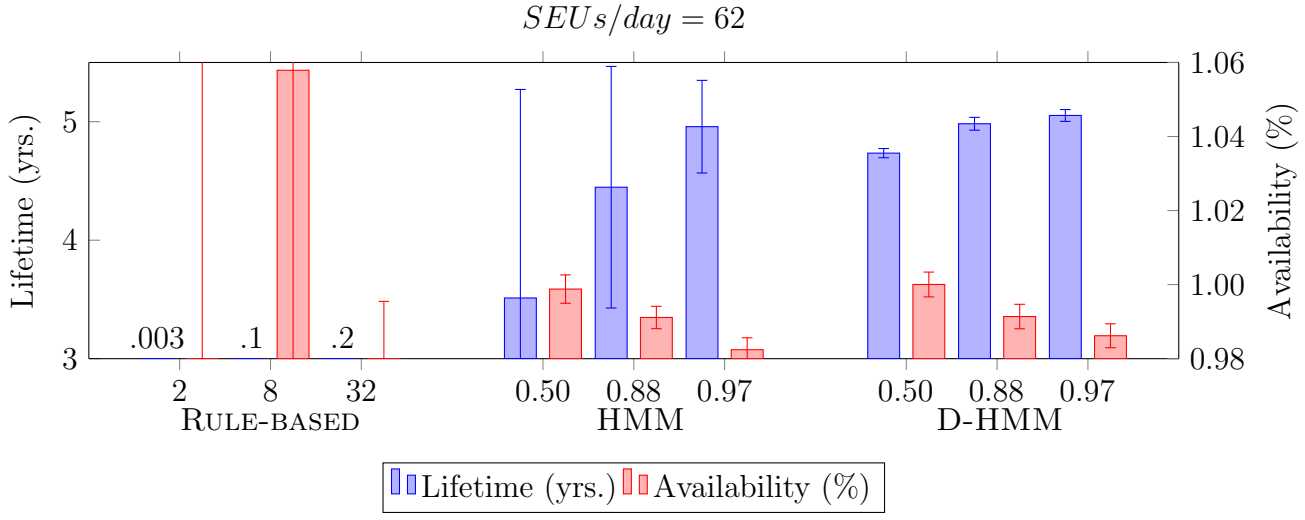



Figure 5.6 Comparison of system lifetimes and normalized availabilities (see Equation 5.9) for the rule-based, HMM-based, and dynamic HMM-based approaches with different thresholds, assuming $N = 10$ resources and a scrubbing period $T = 1$h, when SEUs/day = 62.

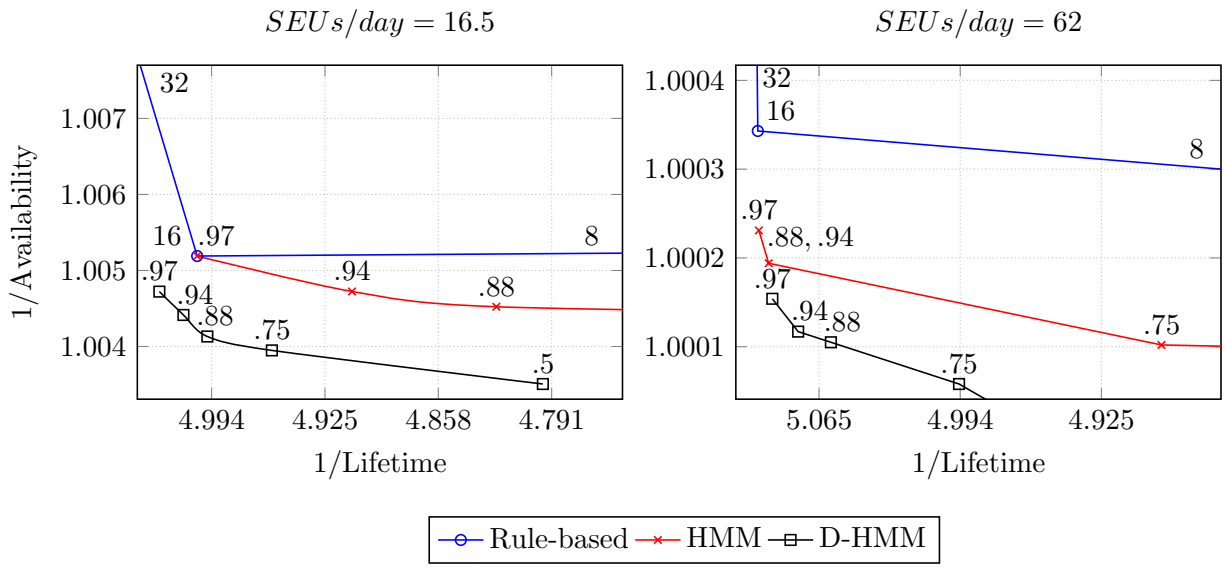



Figure 5.7 Pareto curves obtained by the rule-based, HMM-based, and dynamic HMM-based approaches with different thresholds, assuming $N = 10$ resources and a scrubbing period $T = 10'$, for two levels of SEUs/day.

availability and lifetime metrics. Each point represents the performance of a mechanism (with their associated threshold). In the LEO scenario the points on curve produced by the the proposed D-HMM dominate all other solutions (HMM and rule-based).

Table 5.4 Lifetimes and normalized availabilities (see Equation 5.9) for the rule-based, HMM-based, and dynamic HMM-based approaches with different thresholds, number of resources in the system, and levels of SEUs/day, when the scrubbing period $T = 1$h.

| | | $N = 3$ | | | | $N = 5$ | | | | $N = 10$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $SEUs/day = 16.5$ | | $SEUs/day = 62$ | | $SEUs/day = 16.5$ | | $SEUs/day = 62$ | | $SEUs/day = 16.5$ | | $SEUs/day = 62$ | |
| | thr | L (yrs) | $\widetilde{A}$ (%) | L (yrs) | $\widetilde{A}$ (%) | L (yrs) | $\widetilde{A}$ (%) | L (yrs) | $\widetilde{A}$ (%) | L (yrs) | $\widetilde{A}$ (%) | L (yrs) | $\widetilde{A}$ (%) |
| Rule-based | 2 | 0.00 | 0.943 | 0.00 | 0.589 | 0.00 | 0.903 | 0.00 | 0.823 | 0.01 | 0.986 | 0.00 | 0.925 |
| | 4 | 0.01 | 0.977 | 0.00 | 0.806 | 0.02 | 0.957 | 0.00 | 0.857 | 0.04 | 1.004 | 0.01 | 0.942 |
| | 8 | 0.16 | 1.000 | 0.00 | 0.907 | 0.31 | 0.995 | 0.01 | 0.796 | 0.53 | 0.993 | 0.01 | 1.058 |
| | 16 | 4.74 | 0.997 | 0.01 | 0.916 | 4.79 | 0.998 | 0.02 | 0.876 | 5.00 | 0.995 | 0.04 | 0.983 |
| | 32 | 5.01 | 0.996 | 0.06 | 0.987 | 5.01 | 0.996 | 0.09 | 0.963 | 5.05 | 0.991 | 0.17 | 0.974 |
| HMM | 0.50 | 3.47 | 0.998 | 3.23 | 0.991 | 4.19 | 0.997 | 3.43 | 1.001 | 4.06 | 0.994 | 3.51 | 0.999 |
| | 0.75 | 4.49 | 0.997 | 4.47 | 0.992 | 4.55 | 0.998 | 4.50 | 0.997 | 4.55 | 0.996 | 4.35 | 0.995 |
| | 0.88 | 4.74 | 0.997 | 4.67 | 0.991 | 4.71 | 0.998 | 4.91 | 0.994 | 4.82 | 0.995 | 4.45 | 0.991 |
| | 0.94 | 4.87 | 0.997 | 4.87 | 0.990 | 4.79 | 0.998 | 5.04 | 0.993 | 4.91 | 0.995 | 4.77 | 0.987 |
| | 0.97 | 4.87 | 0.997 | 4.88 | 0.989 | 4.92 | 0.998 | 5.05 | 0.992 | 5.00 | 0.995 | 4.96 | 0.982 |
| D-HMM | 0.50 | 4.75 | 0.998 | 4.73 | 0.996 | 4.80 | 0.998 | 4.77 | 1.002 | 4.80 | 0.997 | 4.73 | 1.000 |
| | 0.75 | 4.89 | 0.998 | 4.92 | 0.995 | 4.92 | 0.998 | 4.89 | 0.999 | 4.96 | 0.996 | 4.91 | 0.995 |
| | 0.88 | 4.93 | 0.998 | 4.96 | 0.994 | 4.96 | 0.998 | 4.95 | 0.998 | 5.00 | 0.996 | 4.98 | 0.991 |
| | 0.94 | 4.97 | 0.998 | 4.99 | 0.993 | 4.98 | 0.998 | 5.00 | 0.996 | 5.01 | 0.996 | 5.03 | 0.988 |
| | 0.97 | 4.99 | 0.998 | 5.02 | 0.993 | 4.99 | 0.998 | 5.04 | 0.995 | 5.03 | 0.995 | 5.05 | 0.986 |

Finally, Table 5.4 summarizes the lifetimes and normalized availabilities (see Equation 5.9) for the rule-based, HMM-based, and D-HMM-based approaches considering all five possible thresholds, and with number of resources ranging from 3 to 10. Data are reported for both the LEO and the HEO, with a scrubbing period $T = 1$ hour. Results show that D-HMM consistently outperforms HMM and rule-based approaches in terms of availability, lifetime, and stability of the results (i.e., lower variance).

## 5.7 Conclusions

In this paper we proposed a failure detection mechanism for electronic components used in space applications that are exposed to both ionizing and particle radiation and, therefore, may experience unpredictable transient faults (SEUs) and permanent faults due to wear out. Our methodology is based on hidden Markov models, however, we extend the classical framework through the use of a dynamic transition model in order to cope with failure time probability distributions that are not memoryless.

We defined lifetime and a normalized availability measure as the two metrics that should be maximized by a properly functioning failure detection mechanism, and we showed that a trade-off between the two is inevitable. We simulated our approach using a simple yet powerful model, performing error injection with realistic parameters, in order to assess its validity. Moreover, we compared the performance of our approach against the performance of a rule-based approach, showing improvements in both the goal metrics and reductions in their variance, and a non-dynamic hidden Markov model approach, showing how the framework benefits from our enhancements.

# CHAPTER 6 ARTICLE 3 – TRADING OFF POWER AND FAULT-TOLERANCE IN REAL-TIME EMBEDDED SYSTEMS

**Preface:** In this third research article, we extend the scope of the probabilistic modelling of on-board data-handling computers with the introduction of real-time and energy consumption requirements. Knowing that the electronics of a spacecraft are affected by transient errors and permanent faults caused by space radiation [115], we now investigate how to manipulate a traditional tuning knob of microprocessors—i.e., frequency and voltage scaling—to improve their fault tolerance. The main insight of this research is that, in multiprocessor systems, the probability distribution of non-masked transient errors depends on both (i) the rate of occurrence of single event upsets and (ii) the utilization levels of the processing elements. As utilization itself is a function of the operating frequency of each core, we are able to discover a trade-off between reliability and power consumption and exploit it to create adaptive fault tolerance in exchange for energy. These results have implications for the design of all those systems with stringent timing and power demands (utilization is an important metric to establish schedulability). In particular, spacecraft with limited or varying power budgets can be designed in such a way that software tasks with different criticality levels are allocated to the appropriate mission phases.

**Full Citation:** J. Panerati and G. Beltrame, "Trading off power and fault-tolerance in real-time embedded systems," *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Montreal, QC, 2015, pp. 1-8.

**DOI:** `https://doi.org/10.1109/AHS.2015.7231175`



**Abstract:** Reliability and fault-tolerance are essential requirements of critical, autonomous computing systems. In this paper, we propose a methodology to quantify, and maximize, the reliability of computation in the presence of transient errors when considering the mapping of real-time tasks on a homogeneous multiprocessor system with voltage and frequency scaling capabilities. As the likelihood of transient errors due to radiation is environment- and component-specific, we use machine learning to estimate the actual fault-rate of the system. Furthermore, we leverage probability theory to define a trade-off between power consumption and fault-tolerance. If a processing element fails, our methodology is able to re-map the ap-

plication, establishing whether the real-time requirements will still be met, and how reliable the new, impaired system will be. Results show that the proposed methodology is able to adjust mapping and operating frequencies in order to maintain a fixed level of reliability for different fault-rates.

## 6.1 Introduction

Reliability is an essential requirement of computing systems that need to operate autonomously in critical situation, often under stringent time constraints. Together with availability and maintainability, it is one of the properties of dependable system. In this work, we define reliability as the property measuring, in probabilistic terms, the ability of a system to perform correctly. A reliable computing system is one that always, or almost always (e.g., with a desired probability), outputs the correct result of its computation, even when faults occur.

This paper proposes a methodology to quantify and optimize the reliability of multiprocessor embedded systems that are subject to transient faults with a constant (or slowly changing) unknown rate. Transient faults are those errors that may, or may not, reappear if the same computation is repeated on the same resources. Bit-flips and soft errors, i.e., incorrect data, caused by the impact of particle radiation on CMOS-based memories and registers are examples of transient faults. Protecting electronics from this sort of faults is especially important in the development of computing systems intended for the space and aerospace industry. This is true for aircraft flying at high altitude through the South Atlantic anomaly [63], as well as satellites or interplanetary missions that cannot benefit from the shielding of Earth's magnetosphere and atmosphere. However, the interest for robustness to transient faults is not limited to aerospace. In fact, as the transistor sizes shrink, less and less energy is required to cause bit-flips in commercial electronics.

## 6.2 Related Work

In this paper, we propose a methodology combining several research areas. Each one of these areas is relevant enough to have its own niche of ongoing research. Therefore, we present the related work in three large categories: research dealing with the mapping of real-time tasks in multiprocessor systems; research dealing with the mitigation of faults, especially those that are induced by radiation; and research dealing with reliability-aware task mapping, scheduling and resource allocation.

While the problem of scheduling real-time tasks on uniprocessor systems is addressed by a number of well established techniques, real-time multiprocessor systems are a lively research

topic. In fact, the optimal mapping—the preliminary phase of each partitioned approach—of applications and tasks is known to be NP-complete [90]. However, Davis and Burns [35], in their survey of real-time scheduling algorithms for multiprocessor systems, note that the use of partitioned approaches allows to exploit a wealth of results on schedulability and optimality for uniprocessor systems. Raravi *et al.* [138], focus on the task-to-processor assignment problem for a specific family of architectures: two-type multiprocessors. They provide three different algorithms with polynomial complexity that can approximately solve of the task-to-processor assignment problem. The algorithm proposed in [12] is based on a linear programming relaxation of the integer linear programming formulation or the task-to-processor assignment, and it guarantees to find a feasible task allocation if the load of each processors is no greater than 50%. In this work, we consider the task mapping problem on multiprocessors systems with distributed frequency scaling capabilities (sometimes treated as heterogeneous systems). However, we only tackle it in the tractable case of a single, multi-tasked real-time application.

Most fault-tolerant systems in the real world rely on simple techniques such as triple modular redundancy and shielding. However, current research on fault-tolerant systems, and especially on systems robust to the effects of radiation, includes approaches implemented at higher design levels. The research of Dumitriu *et al.* [40] and that of Cassano *et al.* [25] make a case for the use of reconfigurable FPGA systems in aerospace applications. Both works introduce macros for the detection and recovery from transient and permanent faults due to radiation effects. In [19] and [115], we find operating system-level approaches to fault-tolerance: the first one based on a trade-off of reliability and performance; the latter on a trade-off of availability and lifetime. Jacobs *et al.* [75] describe a complete framework for dynamically reconfigurable fault-tolerant FPGA systems that also includes a state-of-the-art radiation model, able to describe the time-varying effects of radiation that a spacecraft encounters in a given orbit. The approach we propose in this paper is suitable both to FPGA and OS-level implementations, and specifically targets tolerance to transient faults. Unlike most of the works listed so far, we formalize a way to trade-off reliability not with resources, or weight, or cost, or performance, but with power consumption.

Finally, we consider dependability-aware approaches to the problems of task mapping and scheduling. Qin and Jiang [136] suggest the use of a heuristic approach for the scheduling of real-time tasks on heterogeneous multiprocessor that could also significantly reduce reliability costs. The methodology introduced by Meyer *et al.* [103], on the other hand, is able to find the optimal task mapping and allocation of redundant resources in order to maximize the lifetime of a system. In [20], we also find a task mapping methodology that is able to improve the expected mean time to failure of a system at the cost of an energetic overhead. Self-adaptive

approaches to the task mapping problem based on reinforcement learning have also been proposed [120] The aspect in which our proposal differs from these previous works is the introduction of a probabilistic, quantitative measure of the tolerance to transient faults: we do not just aim at maximizing the expected value of the mean time to failure of the system, but we maximize the probability of the computation to yield the correct result.

## 6.3 System Model

In this section, we define the formalisms that we use to describe the interacting components of our overall system model, i.e. the computing architecture, the software tasks, and the occurrence of transient faults. We also define the power consumption model that we use to estimate the efficiency of each task mapping.

### 6.3.1 Computing Architecture Model

The computing model we consider is that of a homogeneous multiprocessor embedded system or system-on-chip, i.e., a multiprocessor with identical processing elements (PEs) and private caches, as in Figure 6.1. We assume that the performance and power consumption of each PE can be throttled through the use of dynamic voltage and frequency scaling (DVFS). To ensure the best performance and energy savings, the use of voltage and frequency scaling must be coordinated. Each PE will have a number of possible operating points, defined at design time. An operating point consists of a fixed supply voltage and an appropriate frequency step. For example, we could have an energy efficient, low voltage, low frequency operating condition, and several more performing operating conditions, associated to higher voltages and frequencies.

Formally, we will indicate the PEs in an $n$-core multiprocessors as $PE_1, PE_2, .., PE_n$. The operating condition of the $i$-th PE will be written as $OP(PE_i)$: a function that returns a pair $op_j = (f_j, V_j)$, meaning that $PE_i$ is working at the $j$-th operating point and $f_j$ and $V_j$ are its operating frequency and voltage. A system has a fixed, finite number ($\leq k$) of frequency steps and supply voltages, and we indicate the baselines with $f_0$ and $V_0$, respectively.

Computer architectures with such characteristics can be found among commercially available processors, e.g., those in the Intel Core family with Enhanced Intel SpeedStep Technology [71], or could be implemented in a modern FPGA system supporting mixed frequency designs [147], for example the Xilinx Virtex 5 family.

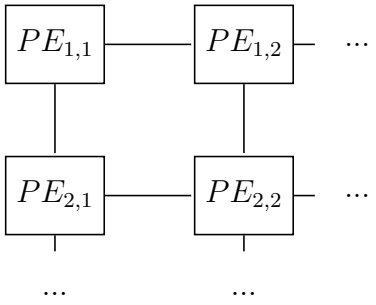



Figure 6.1 A two-dimensional grid architecture with identical PEs and ideal communication links.

### 6.3.2 Real-time Application Model

For the purpose of this work we consider an application as a set of tasks $\tau_1, \tau_2 ..., \tau_m$. Each task has a worst-case execution time (WCET) associated with the slowest operating point of the system $OP(f_0, V_0)$, i.e., the one with the lowest frequency. Because we focus on homogeneous multiprocessor architectures, the WCET is the same on each PE. However, when executing a task on a PE whose operating point has a higher frequency than the baseline frequency $f_0$, we will consider a speed-up proportional to the frequency increase (in Sub-section 6.3.1 we assumed private caches and no shared resources):

$$WCET_{OP(f_i,-)} = WCET_{OP(f_0,-)} \cdot \frac{f_0}{f_i} \tag{6.1}$$

Tasks can also have precedence relationships that we express in the form of a directed acyclic graph (DAG), as in the example in Figure 6.2. An application is executed periodically, with period $T$ that also represents the implicit deadline $D$ by the time its real-time computation has to be completed. However, we cannot determine whether the real-time constraints will be met, or not, without mapping tasks to PEs and fixing the PEs operating points.

As an example, we could consider an application whose goal is to control the attitude of a satellite in order to maximize the amount of energy its solar panels can harvest. The tasks of this application would consist of: determining the satellite position, computing the required adjustment, and actioning its reaction wheels. Because the position of the satellite with respect to the sun might change over time, depending on the orbit, such an application should be executed periodically and in a real-time fashion, i.e., with a bounded delay from the start to the end of its computation.

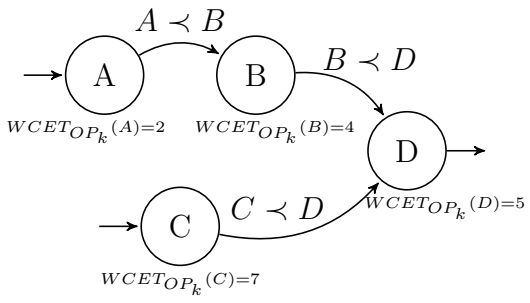



Figure 6.2 The DAG of an application with four tasks. Nodes includes their WCETs at a reference frequency $k$. Arcs express precedence relations.

### 6.3.3 Transient Faults Model

Our analysis focuses on those faults that do not result from the permanent failure of components but are, instead, transient, volatile in nature. Soft-errors and bit-flips are the most common examples of this kind of faults. These faults are often caused by the impact of particle radiation, and they represent a serious hazard for aerospace applications. Because their occurrence arise from the complex interaction of environmental and manufacturing parameters, we choose to model them through the average rate or particle radiation impacts that cause a transient fault. State-of-the-art models of radiation effects, as the one described in [75], in fact, output similar metrics. Our approach, however, is different because we estimate this rate empirically instead of using a predefined physical model.

This rate $\lambda$ can then be used to parametrize a Poisson probability distribution (Figure 6.3) describing the likelihood of the number of transient faults that might occur.

$$P(error) = P(|impacts| \geq 1 \mid |impacts| \sim Pois(\lambda)) \tag{6.2}$$

### 6.3.4 Power Consumption Model

Without considering static dissipation effects such as leakage currents, the switching power dissipated by a CMOS chip is the product of its operating frequency $f$, the square of its supply voltage $V$, its capacitance $C$, and a parameter $\alpha$, called activity factor, that represents the

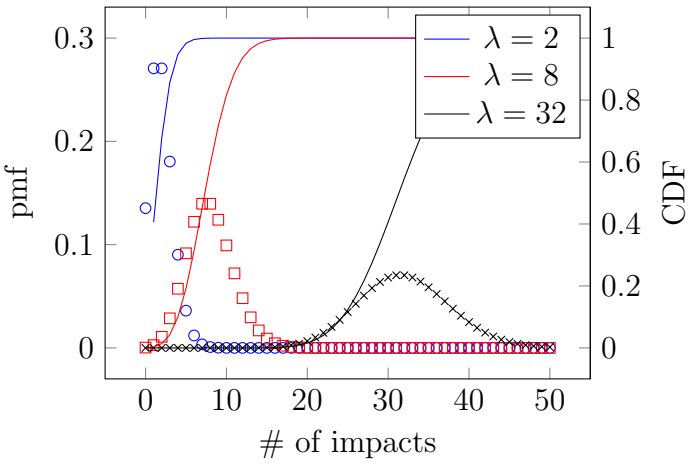



Figure 6.3 Probability mass functions (only marks) and cumulative distribution functions (solid lines) of Poisson distributions with different impact rates $\lambda$.

percentage of gates that actually switch:

$$P = \alpha \cdot C \cdot V^2 \cdot f \tag{6.3}$$

In our multiprocessor model, the total dissipated power will be the sum of the power consumptions associated to each PE. While $f$ and $V$ will change from PE to PE, depending on the operating point (Figure 6.4), the capacitance and the activity factor will not, as we assumed all the PEs to be identical. However, we need to observe that the same computation will take less time on a PE operating at a higher frequency than on a PE with a lower frequency. As a consequence, the total energy consumed by a PE should be scaled by its utilization:

$$E = P \cdot U \cdot T \tag{6.4}$$

where $T$ is the period and implicit deadline of the application described in Subsection 6.3.2, and, by utilization $U$, we mean the fraction of this period in which the PE is actually performing useful computation.

### 6.3.5 Wear Model

Finally, we tackle the problem of how to model the deterioration of our PEs over time. All CMOS-based electronic devices are subject to aging and eventual wear-out through different phenomena, the most significant of which are: hot carriers, negative bias temperature instability (NBTI), time-dependent dielectric breakdown (TDDB), electromigration, and self-heating [171].

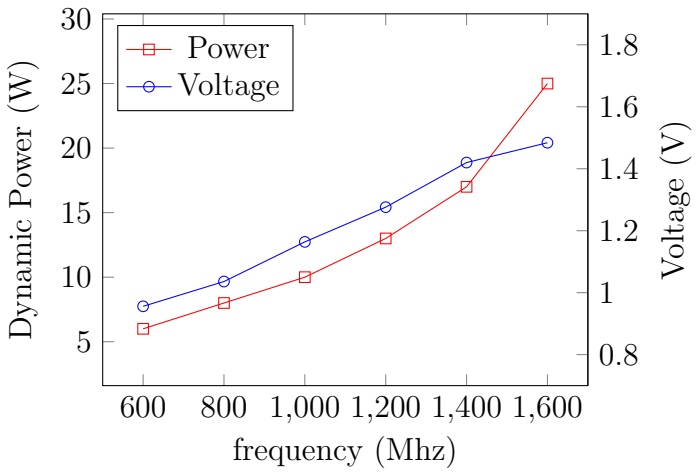



Figure 6.4 The relation between operating frequencies and voltages of the Intel Pentium M processor, and the resulting dissipated power, as reported in [71].

In reliability engineering, the wear-out of electronic components is often approximated using their mean time to failure (MTTF) and a probability distribution. For this purpose, the most commonly used ones are the Weibull, the log-normal, and the exponential distribution. In the latter case, the probability of a failure to occur before time $T$ is:

$$P(f < T) = 1 - e^{\frac{-T}{MTTF}} \tag{6.5}$$

Because exponential distributions are memoryless, the probability of a failure of a functioning PE—in a time interval of fixed length $t$—does not depend on its uptime:

$$P(f < T + t | f > T) = P(f < t) \tag{6.6}$$

Results on commercial avionics systems showed how failures can, in fact, be modeled through and exponential probability distribution, parametrized by the MTTF, as in Figure 6.5.

The modeling of the aging of resources by only using the MTTF and a probability distribution, however, is quite naive, and it lacks the expressive power needed to account for all the aging effects we mentioned above. For example, we know from the Black's equation that the MTTF of a CMOS device has an exponential relation with its temperature [171]. Other causes of aging, e.g., the NBTI, are related to the operating frequencies of the transistors.

Temperatures and voltages are themselves related to the utilization of a PE: a device that is heavily used will reach higher temperatures than a device that is frequently turned off. We modify the traditional probabilistic model of exponentially distributed failures to account for

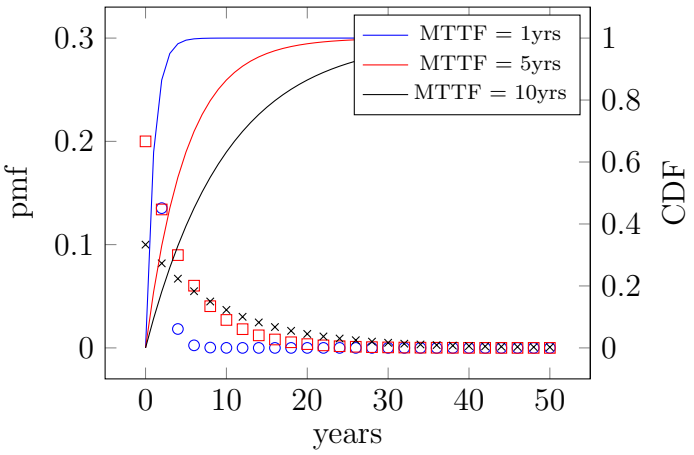



Figure 6.5 Probability mass functions (only marks) and cumulative distribution functions (solid lines) of exponential distributions with different MTTF parameters.

the MTTF variations due to the load of a PE. If the relation between MTTF and utilization was linear, any reduction in the use of a resource would equally impact its expected lifetime, no matter its absolute level of utilization. However, we know that this is not true [103]. Ideally, a device that will never be turned on cannot be proven to have experienced failure. Therefore, in the following, we will assume the existence of an exponential relation the expected lifetime of a PE and its computational load.

$$MTTF_U \propto (MTTF_{100\%})^{U^{-1}} \tag{6.7}$$

Figure 6.6 shows the difference between a linear and an exponential decay of the MTTF with respect to the utilization of a resource.

For PEs that experience variations in their level of utilization over time, we will consider an equivalent utilization as their average utilization weighted by the amount of time spend at each utilization level.

A similar model, despite its simplicity, improves on the traditional probabilistic modeling of failures. With appropriately chosen parameters, such a model would be able to describe the non-linear effects of electromigration on the aging of the PEs.

A notable limitation of this modeling approach is in the way we model the aging of a device subject to a variable level of utilization. In fact, by using a weighted (by the time) average of its utilization levels, we will conclude that a device that is frequently switched on and off will age the same way of a device that is constantly, but mildly, used. We believe that further research should be done on the characterization of the relation between the utilization of

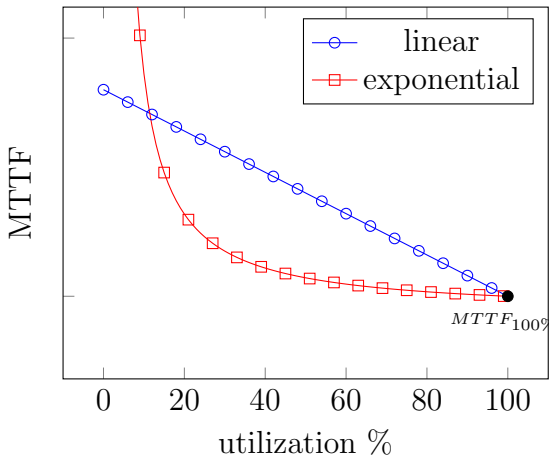



Figure 6.6 Comparison of a linear and and exponential relation between the utilization of a PE and its MTTF. Manufacturer, like Intel, do not do not always disclose failure rates of their CPUs. We assume to know the MTTF of a fully utilized PE, $MTTF_{100\%}$.

PE and its failure probability. More information should also be disclosed by manufacturers, because an online estimation of this relation—as the one we propose for the occurrence of transient faults later in this article—is not possible.

## 6.4 Methodology

The methodology we propose starts with an overview of the task mapping problem on a homogeneous multiprocessor system with DVFS capabilities, it proceeds with a probabilistic formulation of the occurrence of transient faults, and it concludes with the formulas allowing us to compute how task mapping and frequency scaling impact the fault-tolerance and the power consumption of the system.

### 6.4.1 Task Mapping and Real-time Constraints

Given a multiprocessor with $n$ identical PEs and an application composed by $m$ tasks, the number of possible task mappings—a set of functions $q$'s returning the PE assigned to each task, that we will indicate with $Q$—grows exponentially with the number of tasks:

$$|M| = n^m \tag{6.8}$$

However, it is important to observe that, under assumption of identical PEs, the number of indistinguishable mappings is much smaller. In fact, the number of distinct partitionings of a set of size $m$ is known as the Bell number [141], and it can be recursively computed

as $B_n = \sum_{k=0}^{n-1} B_k \binom{n-1}{k}$. For example, if $n = 3$ and $m = 6$, $3^6 = 729$ but $B_6$ is only 203. Moreover, in most practical cases, $n < m$ which implies that some of the partitionings accounted by the Bell number would be unfeasible, and their actual number would be given, instead, by the sum of the Stirling numbers of the second kind [168] up to the number of PEs:

$$|M| = \sum_{i=1}^{n} S(m, i) \tag{6.9}$$

Having found the number of distinct task mappings, we observe that, if each PE performs DVFS independently, and we have $k$ different operating points for each PE, the size of our design space—a set $S$ whose elements are couples $(q, \mathbf{op})$ where $q \in Q$ and $\mathbf{op}$ is a vector of operating points, one for each PE—will also grow exponentially with $k$:

$$|S| = n^k \cdot \sum_{i=1}^{n} S(m, i) \tag{6.10}$$

In order to prune down this large search space, we need to focus on the relation between the task mapping, the operating point of each PE, and the real-time constraints of the application. To know whether an application will meet its deadline or not, we need to compute its overall execution time. Because our application is expressed by a DAG of tasks, this execution time corresponds to the longest path of the DAG—a linear complexity problem—where each arc has a weight equal to the WCET of its antecedent task node.

An efficient way to explore our design space would consist of evaluating each task mapping with all the PEs in their most performing operating point (in Section 6.3 we assumed speed-ups proportional to the variations in frequency). If an application can be scheduled within its deadline, then we can examine lower frequency operating points. However, as soon as the deadline is not met, we will be able to discard all the points in the design space that assign an equal or slower operating point to that PE.

### 6.4.2 Utilization Levels

Once the design space has been pruned of all the design points with indistinguishable mappings or whose vectors of operating points do not allow to meet the real-time constraints, we can compute the utilization level of each PE. We will use $U_i$ for the utilization of the $i$-th PE. This value will be equal to the sum of the WCET of the tasks mapped to the $i$-th PE, divided by period of the application $T$.

$$U_i = \frac{\sum_{j|q(\tau_j)=PE_i} WCET_j}{T} \tag{6.11}$$

A vector of utilization levels—one for each PE—can be unequivocally computed, given the DAG of an application, its period, the task mapping, and the operating point of all PEs.

### 6.4.3 Particle Radiation and Transient Errors

We now tackle the problem of modeling the errors arising from transient faults due to the impact of particle radiation on our multiprocessor system. We assume that these impacts—and, therefore, the transient faults—occur at a constant but unknown rate $r$. In a preliminary learning phase, all we can observe is the number of erroneous computations produced by a training application whose task mapping, operating points, and utilizations are known. We will indicate the fraction of erroneous results, out of all the learning computation, with $o$. This value is not the same as $r$ because not all transient faults result in errors, but they might be masked if they arise in a PE while it is not performing useful computation.

First of all, we compute the number of particle impacts/transient faults on each of the $n$ PEs during the execution period $T$. Assuming that the chip is hit by radiation uniformly, this number is $(T \cdot r)/n$. However, the $i$-th PE is active only during a fraction of $T$, equal to its utilization. Therefore, the number of transient faults, unmasked by inactivity, is:

$$\lambda_i = U_i \cdot \frac{r \cdot T}{n} \tag{6.12}$$

This value can be used to parametrize a Poisson probability distribution and compute the probability observing an error in the $i$-th PE, i.e., the probability of one or more unmasked transient faults $UTF$ over the period $T$:

$$\begin{aligned} P_{PE_i}(error) &= P(UTF \geq 1 | UTF \sim Pois(\lambda_i)) \\ &= 1 - P(UTF = 0 | UTF \sim Pois(\lambda_i)) \\ &= 1 - e^{-\lambda_i} \end{aligned} \tag{6.13}$$

At this point we can compute the probability of observing a system-wide error due to an unmasked transient fault, as the probability of an error happening in at least one PE. Assuming that errors happen in different PEs independently, this is $P_{system}(error) = 1 - \prod_{i=1}^{n} e^{-\lambda_i}$, or, simply:

$$P_{system}(error) = 1 - e^{\sum_{i=1}^{n} -\lambda_i} \tag{6.14}$$

However, this probability is approximated by the value $o$ that we computed in our preliminary learning phase. Therefore, reversing Equation 6.14, we can finally compute an estimate of $r$,

the constant impact rate.

$$r = \frac{n \cdot (1 - \ln(o))}{T} \cdot \sum_{i=1}^{n} -U_i \tag{6.15}$$

### 6.4.4 Fault-tolerance Optimization

There is an important observation to make about Equation 6.14: in order to minimize the probability of observing an unmasked transient fault resulting in a system-wide error, we need to minimize $\sum_{i=1}^{n} -\lambda_i$. However, from Equation 6.12, we know that $\lambda_i$'s are proportional to system-wide parameters—$r$, $T$, and $n$—and the utilization of each PE. Therefore, the way to minimize:

$$\sum_{i=1}^{n} -\lambda_i = \frac{r \cdot T}{n} \cdot \sum_{i=1}^{n} U_i \tag{6.16}$$

is to find the operating points with the lowest average utilization level. This can be done by choosing those operating points having the highest frequencies on each PE that is loaded with at least one task. This is a fairly intuitive result: minimizing the time we spend on computation will minimize the probability of being affected by transient errors occurring at a constant rate. What is new here, is the formulation of a probability measure of the reliability of the computation. In fact, by plugging $r$ (computed after the learning phase through Equation 6.15) and the utilizations (computed from the operating points assigned to each PE) in Equation 6.14, we can compute the likelihood of getting an erroneous—or, viceversa, correct—result. This estimate will still be slightly pessimistic, as, in complex applications, errors might also be masked by the specific type of computation performed.

### 6.4.5 Power Consumption Optimization

Combining Equations 6.3 and 6.4, we can express the total energy consumption of the $i$-th PE, working at the $j$-th operating point, over one execution period $T$ as:

$$E_i = \alpha \cdot C \cdot V_j^2 \cdot f_j \cdot U_i \cdot T \tag{6.17}$$

The activity factor $\alpha$, the capacitance $C$, and the period $T$ do not vary from one PE to another. It is important to observe that the frequency factor $f_j$ and the utilization factor $U_i$ will cancel each other out, because of the speed-up assumed in Equation 6.1. Therefore, in order to obtain energy savings, we will need to act on the supply voltage. Because of the quadratic relation in Equation 6.17, a small tuning of the supply voltage would result in a largely improved energy efficiency. These savings will be even higher if we consider that high performance operating points also have higher leakage currents.

### 6.4.6 The Power and Fault-tolerance Trade-off

The work presented so far shows that by tuning the operating points of our system we can influence both its fault-tolerance and its power consumption. Frequencies and supply voltages are not independent—their specific relation depends on the operating points allowed by the manufacturer—but, usually, move in the same direction.

However, while tuning voltages impacts the energy budget quadratically, tuning frequencies impacts on utilizations linearly. Utilizations, in turn, reflect exponentially on the probability of a system-wide error. Whereas the lower energy consumption the better, for reliability purposes, we will only want the probability of an error to be below a certain threshold. These considerations make the case for the quantitative evaluation of the trade-off between fault-tolerance and power consumption. In Section 6.5, we show how this works out with parameters from the real world.

### 6.4.7 Lifetime Optimization

We observe from Equation 6.14 that the probability of the occurrence of system-wide errors due to the impact or particle radiation essentially depends on the average utilization of the PEs. This means that, if we only look at the fault-tolerance of the system, we might find design points with different task mappings and operating points but that are indistinguishable from a reliability perspective.

However, in the model we described in Section 6.3, changing the computational load distribution of the system affects its performance and, in particular, its expected lifetime. Therefore, we want to find, among the design point with the highest fault-tolerance, those whose computational load distribution (i.e., the utilizations of their PEs) is such that their expected lifetime is the longest.

One problem that we encounter is the definition of the utilization of a PE that is subject to a variable load. In our model we made the assumption that the equivalent utilization $\hat{U}$ of this PE is its average utilization weighted by the amount of time spent at each utilization level:

$$\hat{U} = \frac{1}{\sum_{i=1}^{l} \Delta t_i} \sum_{i=1}^{l} \Delta t_i \cdot U(\Delta t_i) \tag{6.18}$$

In Equation 6.18, we discretized the time in $l$ steps $\Delta t$, and represented the utilization level of a PE during the $i$-th step as $U(\Delta t_i)$. First, we establish the expected time of failure of the first PE to fail. As we assumed an exponential relation (but it might be any other suitable function, e.g., a fitted polynomial) between utilization and MTTF, and the initial utilization

level of each PE $U_i(\Delta t_0)$ are known, this value will be equal to:

$$MTTF_{1st} = \min_{1 \leq i \leq n} (MTTF_{100\%})^{U_i^{-1}} \tag{6.19}$$

After the failure of the first resource, a new design point for the new $n-1$ multiprocessor will be chosen, keeping in mind fault-tolerance, power, and real-time requirements, according to the same methodology we previously exposed. New utilization levels will be computed for each PE, and the expected time of failure of the second resource to fail will be computed as:

$$MTTF_{2nd} = \min_{1 \leq i \leq n} (MTTF_{100\%})^{\frac{\Delta t_1 + \Delta t_2}{\Delta t_1 \cdot U(\Delta t_1) + \Delta t_2 \cdot U(\Delta t_2)}} \tag{6.20}$$

Depending on the type of relation that we assumed between utilization and MTTF, this second time of failure, might be have a closed form solution or not. In our case, the exponential relation assumption, leads us to the solution of an equation having the form of $x^x = k$, or $\log x = \frac{1}{a \cdot x + b}$. In other words, we want to find the first quadrant intersection of the logarithm with base $MTTF_{100\%}$ an the hyperbola having the weighted average utilization as the denominator:

$$\log_{MTTF_{100\%}}(MTTF_{2nd}) = \frac{\Delta t_1 + \Delta t_2}{\Delta t_1 \cdot U(\Delta t_1) + \Delta t_2 \cdot U(\Delta t_2)} \tag{6.21}$$

where $\Delta t_1 = MTTF_{1st}$ and $\Delta t_1 + \Delta t_2 = MTTF_{1st}$. This can be done iteratively using several methods. As we know that the solution will be bounded to have an $x$ between 0 and 1, we use a binary search and progressively refine the estimate of the solution. Another possible approach (non exploited here, but that might converge more quickly to a solution) is Newton's method, that improves an initial guess $x_0$ through the formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{6.22}$$

Then, this approach can be reused to compute the expected failure time of each one of the PEs of the system and, eventually, its death. One important consideration to make is that, in most practical scenarios, the impossibility to meet real-time constraints will arise before the complete death of the system. With the decrease of available PEs, tasks might not be scheduled within their deadline. We have to distinguish, then, between the "total lifetime" of a system, i.e., the time that will pass before the failure of its last PE, and the "functional lifetime" of the system, i.e., the time that will pass before resources become to scarce to meet the real-time constraints.

Whether any of this two metrics will be improved by a homogeneous distribution of the

computational load over the resources, or by an uneven utilization, it is strictly related to the nature of the task set we are optimizing for, and the shape of relation that we assumed between the utilization and the MTTF of the PEs. In the next section, we show how this works out with parameters from the real world.

### 6.5 Case Study

For the sake of simplicity, we map an application with four tasks ($m = 4$) on a dual-core ($n = 2$) architecture with three operating points available on each PE: $OP_1$ = (1.6GHz,1.484V), $OP_2$ = (1.2GHz,1.420V), $OP_3$ = (600MHz,0.956V), i.e., a sub-set of the six operating points of the Intel Pentium M processor[71]. Reported power consumptions are 25W, 13W, and 6W, respectively. The real-time application has a period $T$ equal to its deadline $D$ of 10 seconds and two precedence relations: $A \prec B$ and $C \prec D$. The WCETs of the each task in each operating point are reported in Table 6.1. The size of our case study is limited for display purposes only. As we explained in Section 6.4, the only issue with the scalability of our approach is in the number of operating points that yields an exponential—but easily prunable—growth of the design space.

While $n^m = 16$ and the Bell number of four is $B_4 = 15$ , the sum of the Sterling numbers $S(4, 1) = 1$ and $S(4, 2) = 7$ tells us that there are only 8 distinct partitionings. Multiplying by $k^n = 9$, we find 72 possible design points. After the pruning of all those points that do not allow to meet the real-time constraints, we are left with 29 acceptable design points. We observe that the magnitude of this pruning phase is really dependent on the strictness of $T$. For $T = 12$, we would have as many as 51 acceptable points.

Now, we imagine our system to be deployed on a satellite that might find itself in a Low Earth Orbit (LEO) or in a Highly Elliptical Orbit (HEO). In these conditions, we know that the number of particle radiation impacts, per day, on a Virtex 4 board, is 16.5 and 62 respectively [75]. From Equation 6.14, we can evince that the probability of a system-wide error is a function of its average utilization. Therefore, in Table 6.2, we report the seven levels of average utilization that can be achieved in the 29 acceptable design points. For each one of them, we report the lowest achievable power consumption (power consumption also depends on the load distribution), and the system reliability in terms of number of error over 10000 computations—for a $T$ of 10 seconds, it means slightly more than a day—both in LEO and HEO. In Figure 6.7, we map the acceptable design points to an $n$-dimensional design space, where each dimension represents the utilization level of a PE. Only 15 points appear in Figure 6.7 because, due to the limited size of this particular case study, several of the original 29 have identical utilizations.

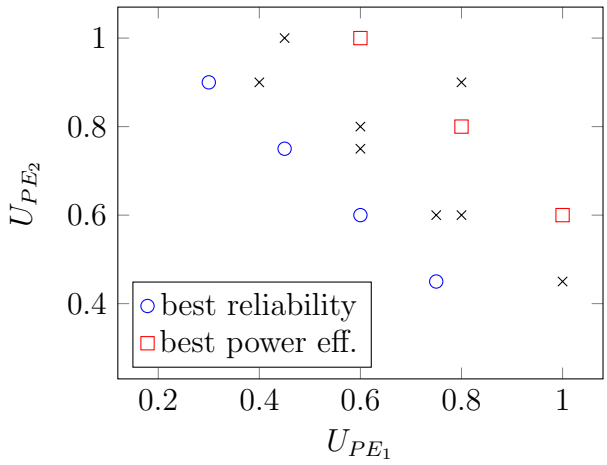



Figure 6.7 Design space as an $n$-dimensional space of utilization levels, with ideal reliability and power consumption design points.

As we expected, increasing the frequencies of the PEs and reducing their utilization allows to mask errors. In our practical example, reducing the average utilization by 25% results in $\sim$ 25% less errors both in LEO and HEO, at the cost of increasing the power consumption by 50%. However, it is important to note that the relation between the average utilization and the number of errors is not linear (Equation 6.14) as it might seem from these numbers: in environments with higher particle impact rates, we would need a larger reduction in the average utilization to have similar gains in reliability.

## 6.6 Conclusions And Future Works

In this paper we proposed a methodology for the scheduling of real-time task in homogeneous multiprocessor systems that is both reliability- and energy-aware. Notably, we used probability theory to quantify the reliability of the system to transient faults—occurring at a

Table 6.1 WCETs (in seconds) for each one of the four tasks, in each of the three operating points.

| | | Operating Point | | |
|---|---|---|---|---|
| | | $OP_1$ | $OP_2$ | $OP_3$ |
| | | $f_1 = 600$MHz | $f_2 = 1.2$Ghz | $f_3 = 1.6$Ghz |
| Task | A | 8.0 | 4.0 | 3.0 |
| | B | 4.0 | 2.0 | 1.5 |
| | C | 8.0 | 4.0 | 3.0 |
| | D | 12.0 | 6.0 | 4.5 |

Table 6.2 Reliability and power consumption metrics for different design points, ordered by the average utilization of the PEs. The period $T$ is 10s, the number of errors is over 10000 computations.

| Average Utilization | Best Power Consumption | System Errors LEO | HEO |
|---|---|---|---|
| **0.600** | 30.00W | **12** | **42** |
| 0.650 | 27.70W | 13 | 45 |
| 0.675 | 26.55W | 14 | 47 |
| 0.700 | 25.40W | 15 | 49 |
| 0.725 | 24.25W | 15 | 50 |
| 0.800 | **20.80W** | 16 | 56 |
| 0.850 | 27.30W | 17 | 59 |

fixed, unknown rate—with respect to the task mapping and the PE utilization levels. We also showed how to improve this probabilistic reliability measure at the cost of a higher energy expense.

Our future research focus includes the development of this work in two directions: on one hand, we aim at the implementation and physical testing of our approach on an FPGA system; on the other hand, we want to extend the probabilistic framework described in this work to include the effects of interconnects and permanent faults. In particular, we think that, by taking into account the overheating and the wearing of PEs with high levels of utilization, we could also improve the overall system lifetime.

# CHAPTER 7 ARTICLE 4 – ASSESSING THE RESILIENCE OF STOCHASTIC DYNAMIC SYSTEMS UNDER PARTIAL OBSERVABILITY

**Preface:** During a stay at the Inoue Laboratory of the National Institute of Informatics in Tokyo, I broadened the scope of my research and started investigating a more general category of adaptive systems. This chapter addresses the challenges posed by the definition, modelling, and design of resilient ecosystems and environments. Resilience is recognized as a property of interest for the analysis of many complex systems but a unique definition of it does not yet exist. Typically, a system is considered to be resilient if it can adjust in response to shocks and still provide the services it was intended for. Starting from a definition of resilience proposed in the context of constraint-based systems, we identify an efficient exact algorithm that computes the essential inference steps required to assess the resilience of stochastic systems that can only be observed through imperfect measurements. To demonstrate the wide applicability of this research, we describe how to use it to draw insights and improve decision making in two scenarios that are related to computer engineering (self-aware computing and swarm robotics) and two that are not (disaster management and macroeconomics). As the main author of the article, my contributions included: proposing a HMM-based framework, identifying the inference algorithm, studying its theoretical complexity, writing the code, detailing the example scenarios, performing the experiments and simulations, and writing a first draft of the document.

**Authors:** Jacopo Panerati, Nicolas Schwind, Stefan Zeltner, Katsumi Inoue, and Giovanni Beltrame

**Submitted To:** Science Advances

**Abstract:** Resilience is a property of major interest for the design and analysis of generic complex systems. A system is resilient if it can adjust in response to disruptive shocks, and still provide the services it was designed for, without interruptions. In this work, we adapt a formal definition of resilience for constraint-based systems to a probabilistic framework derived from hidden Markov models. This allows us to more realistically model the stochastic evolution and partial observability of many complex real-world environments. Within this framework, we propose an efficient and exact algorithm for the inference queries required to construct generic property checking. We show that the time complexity of this algorithm is on par with other state-of-the-art inference queries for similar frameworks (that is, linear with

respect to the time horizon). We also provide considerations on the specific complexity of the probabilistic checking of resilience and its connected properties, with particular focus on resistance. To demonstrate the flexibility of our approach and to evaluate its performance, we examine it in four qualitative and quantitative example scenarios: (1) disaster management and damage assessment; (2) macroeconomics; (3) self-aware, reconfigurable computing for aerospace applications; and (4) connectivity maintenance in robotic swarms.

## 7.1 Introduction

Originally coined in the context of environmental sciences and ecological systems, resilience has become a property of great interest for the study of complex systems. Although resilience is not easily defined, researchers agree that it is a fundamental characteristic of those ecosystems that are able to absorb extreme spikes and survive, albeit transformed. The insect populations of North-eastern American forests [67] are well-known examples of such resilient systems.

The focus of the artificial intelligence community has been, so far, on narrowing down the concept of resilience and formalizing it, for example in constraint-based and non-deterministic dynamic systems [149]. These approaches are extremely general and able to describe a plethora of real-world systems, but they have very limited predictive power. The transition models in non-deterministic dynamic systems resemble those of Markov chains and decision processes but, because they do not have probability distributions associated to transitions, they do not tell whether a future world is more likely than the others.

Succeeding in the definition and implementation of resilience has the potential to enable the creation of "resilient by design" systems. In computing engineering, for example, networks and robotics systems provided with resilient properties will possess the ability to absorb shocks and to transform in response to external attacks, while still providing their services.

By choosing to study resilience in the context of hidden Markov models, we extend the existing artificial intelligence research to take into account the unpredictability of the real world. This is essential to make our model consistent with the idea of a "random world" proposed by Holling [67]. In fact, conditional probability distributions can be seen as the stochastic extension of non-deterministic transition functions.

In this work, we also expand the previous discussion about resilience with the element of partial observability, adding one more layer of complexity. In the end, the goal of our work is to provide the formal and algorithmic tools to efficiently answer queries such as: "what is the likelihood of requiring extra personnel in an emergency area over the next three days?",

"what is the probability that a worker robot will soon become disconnected from its assigned cluster?" or "with 99% confidence, what is the minimal number of neighbor links to maintain connectivity in an extremely noisy network?".

## 7.2 Related Work

In a seminal paper from 1973, Holling introduced the concept of "resilience of ecological systems" [67]. In it, he draws a clear separation line between resilience and the more commonly used notion of stability. Resilient systems are not those systems that simply react to imbalances by quickly returning to equilibria. Instead, when perturbed, they are able to find new sustainable configurations. It is worth noting that Holling defines resilience in the context of what he calls "the random world": an environment that is intrinsically stochastic. Developing these ideas, Walker *et al.* [169] define resilience as "the capacity of a system to absorb disturbance and reorganize while undergoing change so as to still retain essentially the same function, structure, identity, and feedbacks".

Computer science has often looked at biology as a source of inspiration for the development of search algorithms, coordination mechanisms, and complex frameworks. The first attempt to develop a formal definition of resilience exploiting the tools of artificial intelligence was provided by Överen, Willsky, and Antsaklis [110] and, successively, further developed by Baral *et al.* [10] and Schwind *et al.* [149]. Our research is based on the formal description of the System Resilience- (SR-)model introduced by Schwind *et al.* [149]. When compared to the existing research [149, 10], the main distinctive trait of our work is in its integration of the ideas of probability theory. Our analysis of resilience is based on the probabilistic framework of hidden Markov models (HMMs). HMMs are often employed in applications such as signal and natural language processing. Nonetheless, they have also proven to be fruitful descriptive tools for many other complex dynamic systems [137].

Our methodology is closely connected to the sub-field of artificial intelligence that deals with probabilistic graphical models, dynamic Bayesian networks in particular. The two most common types of inference tasks for probabilistic graphical models—the larger family of frameworks HMMs belong to— are marginal and maximum a posteriori estimation (i.e., the computation of the distribution of a single variable and the most likely assignment of all variables, respectively). For these, efficient algorithms with convenient linear time-complexity have been identified [143]. In the following and Methods sections, we show that these queries are not sufficient to perform the kind of property checking demanded by our formal definition of resilience. An *ad hoc*, efficient algorithm to answer the necessary queries is detailed in the Methods section.

With regard to applications, resilience has been, in recent years, a topic of interest for researchers in many different areas. Beyond ecology, these areas include economics, networking, critical and real-time systems, and swarm robotics—a domain that lies at the prolific intersection of computer engineering and biology. Researchers have been developing ways to formalize the robustness and resilience [179, 146] of networks of robots with respect to their most common tasks, e.g., consensus, flocking, and formation. Our work shares some terminology with this research and can also be used to address fundamental problems of swarm robotics (e.g., the one of connectivity). However, it is worth noting that the formal definition of resilience given here is not a domain-specific one and it could be used orthogonally with that, for example, of Saldaña *et al.*[146] (see the Application Scenarios section).

### 7.3 Resilience and Resilient Properties in Probabilistic Models

This work re-interprets a formal definition of resilience (for dynamic systems) [149] using the probabilistic framework of hidden Markov models and enriching it with a cost function. In this section, we recall and combine together a number of definitions that are derived from recent research work on formal resilience in dynamic non-deterministic constraint-based models [149, 148] and timed probabilistic models [118].

The SR-model is a theoretical framework proposed by Schwind *et al.* [149] that combines elements of constraint-based systems and non-deterministic dynamic systems. It gives us a formal definition of resilience, as the unifying property arising from three simpler properties: 1) resistance, 2) functionality, and 3) recoverability. The SR-model consists of two separate formal descriptions for the kinematics and the dynamics of a system: the first is represented as sequences of pairs called "state trajectories" or SSTs:

$$SST = (CBS_0, \varsigma_0), (CBS_1, \varsigma_1), \ldots, (CBS_i, \varsigma_i), \ldots \tag{7.1}$$

The subscript index skims through the time steps. The symbol $CBS_i$ represents a constraint-based system composed of a set of variables $\mathbf{X_i}$ and a cost function $\kappa_i$:

$$CBS_i = \langle \mathbf{X_i} = \{X_i^0, X_i^1, \ldots, X_i^j, \ldots\}, \kappa_i : \mathcal{D}(\mathbf{X_i}) \to \mathbb{R}^+ \rangle \tag{7.2}$$

The second element in each pair, $\varsigma_i \in \mathcal{D}(\mathbf{X_i})$, represents a complete assignment of the variables in $\mathbf{X_i}$: $\varsigma_i \in \mathbb{R}^{|\mathbf{X_i}|}$. Each SST corresponds unambiguously to a sequence of costs obtained by plugging-in each $\varsigma_i$ into its corresponding cost function $\kappa_i$: $\kappa_0(\varsigma_0), \kappa_1(\varsigma_1), \ldots$. The envi-

ronment dynamics are described using non-deterministic Dynamic Systems (DSs):

$$DS = \langle \mathbf{CBS}, \mathbf{A}, m : \mathbf{CBS} \times \mathbf{A} \rightarrow \mathcal{P}(\mathbf{CBS}) \rangle \tag{7.3}$$

where **CBS** represents the set of all possible constraint-based systems $CBS_i$, **A** is the set of actions available at each time step, and $m$ is a non-deterministic transition function that, given the current CBS and an action, returns the set of possible constraint-based systems for the next time step.

The kinematic description of the SR-model (SSTs and sequences of costs) is central to the formalization of resilience and it is preserved in our proposed methodology. However, we prefer to discard the non-deterministic description of the dynamics in favor of a probabilistic approach based on hidden Markov models. Hidden Markov models (HMMs) can be seen as specific subset of both dynamic Bayesian networks (DBNs) and state-observation models [143]. HMMs have a single discrete state variable $S$ and a single discrete observation variable $O$. A HMM is fully specified by the probability distribution of $S$ at time $-1$, $P(S_{-1})$, the conditional distribution of $O$ given $S$ at the same time step, $P(O_t \mid S_t)$, and the conditional distribution of $S$ given $S$ at the previous time step, $P(S_{t+1} \mid S_t)$ [86].

$$HMM = \langle P(S_{-1}), P(O_t \mid S_t), P(S_{t+1} \mid S_t) \rangle \tag{7.4}$$

HMMs are commonly used for the tasks of signal processing and speech recognition [86] because efficient (i.e., with computational time complexity that is linear with respect to the time horizon of the model) algorithms exist for: 1) the estimation of the probability distribution of $S$, also called the "hidden" variable, taking only assignments of $O$ as input (filtering and smoothing algorithms); and 2) the identification of the most likely sequence of assignments of $S$.

To formalize resilient properties in the probabilistic context of a "random world", HMMs offer the probabilistic reasoning of DBNs and the independence assumptions of state-observation models. We chose HMMs above other frameworks such as Markov decision processes (MDPs) and partially-observable Markov decision processes (POMDPs) because the additional complexity of their decision layer was deemed unnecessary for the sole assessment of resilience.

The creation of a new framework to describe the resilience of stochastic, partially observable systems, requires, however, certain additional steps. First, we re-define the domain of the random variables $S$ and $O$ as the union of the domains of the set of variables of the constraint-based systems in **CBS**:

$$\Omega(O) \subseteq \Omega(S) = \cup_i \, \mathcal{D}(\mathbf{X}_i) \tag{7.5}$$

Without loss of generality, we impose a static cost function: $\forall i,\ \kappa_i = c : \Omega(S) \rightarrow \mathbb{R}^+$ and we introduce a sensor model that describes the imperfect observations of the set of variables: $P(O_t \mid S_t) : \cup_i\ \mathcal{D}(\mathbf{X}_i) \times \cup_i\ \mathcal{D}(\mathbf{X}_i) \rightarrow [0, 1]$. Because we are not interested in formulating a decision making problem, we drop the set of actions $\mathbf{A}$ from $DS$ and we replace $m$ with the conditional probability distribution that describes the probability of a set of variables evolving into another:

$$P(S_{t+1} \mid S_t) : \cup_i\ \mathcal{D}(\mathbf{X}_i) \times \cup_i\ \mathcal{D}(\mathbf{X}_i) \rightarrow [0, 1] \tag{7.6}$$

Putting these elements together with an initial probability distribution $P(S_0) : \cup_i\ \mathcal{D}(\mathbf{X}_i) \rightarrow [0, 1]$, our overall framework (shown in Figure 7.1) can be re-written as:

$$c\text{-}HMM = \langle P(S_{-1}), P(O_t \mid S_t), P(S_{t+1} \mid S_t), c : \Omega(S) \rightarrow \mathbb{R}^+ \rangle \tag{7.7}$$

In the SR-model, resilience is a boolean property of a state trajectories SST. It can be seen as a unifying property, combining different desirable behaviours of a dynamic system and arising from three simpler properties of state trajectories: resistance, functionality, and recoverability (see Figure 7.2).

***l*-resistance** The resistance property expresses the fact that a trajectory never incurs in a cost that is larger than a fixed threshold. Therefore, this property is parameterized by this maximum acceptable cost.

**Definition 1.** *Given a state trajectory* $SST = (CBS_0, \varsigma_0), (CBS_1, \varsigma_1), \ldots$ *and a positive threshold* $l \in \mathbb{R}^+$*, SST is said to be l-resistant if and only if each cost in its corresponding*

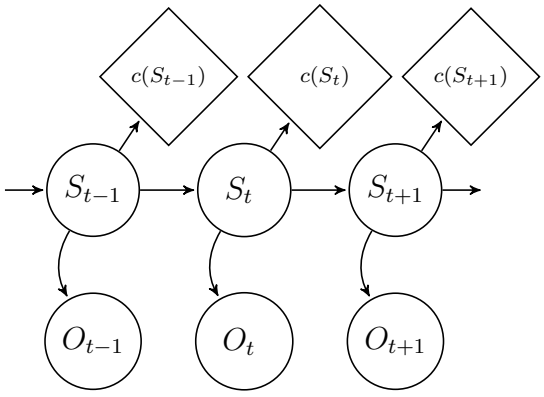



Figure 7.1 Unrolling of the C-HMM framework over three time steps.

*cost sequence is less than or equal to the threshold $l$:*

$$\kappa_i(\varsigma_i) \leq l \quad \forall \kappa_i(\varsigma_i) \in (\kappa_0(\varsigma_0), \kappa_1(\varsigma_1), \dots, \kappa_n(\varsigma_n), \dots) \tag{7.8}$$

This property must be satisfied whenever we deal with periodic, fixed budgets.

**$f$-functionality** The functionality property tells us if the costs of a trajectory are, on average, equal to or below a certain threshold. As in the case of resistance, this threshold parameterizes the property.

**Definition 2.** *Given a state trajectory $SST = (CBS_0, \varsigma_0), (CBS_1, \varsigma_1), \dots$ and a positive threshold $f \in \mathbb{R}^+$, $SST$ is said to be $f$-functional if and only if the arithmetic average of the costs in its corresponding cost sequence is less than or equal to the threshold $f$:*

$$|SST|^{-1} \cdot \sum_{i=0}^{|SST|} \kappa_i(\varsigma_i) \leq f \tag{7.9}$$

This property is important when the operations we plan and our budget have different time granularity.

**$\langle p, q \rangle$-recoverability** The recoverability property concerns those systems in which costs over a certain threshold can be accepted, but only as long as the system is able to return within normal conditions before consuming a fixed, restorable, reserve.

**Definition 3.** *Given a state trajectory $SST = (CBS_0, \varsigma_0), (CBS_1, \varsigma_1), \dots,$ a positive threshold $p \in \mathbb{R}^+$ and a positive budget $q \in \mathbb{R}^+$, $SST$ is said to be $\langle p, q \rangle$-recoverable if and only if every time the sequence of costs exceeds the threshold, it also returns below (or at) it before the cumulative offset surpasses the reserve:*

$$\forall k \ s.t. \ \kappa_k(\varsigma_k) > p, \exists j > k \ s.t. \ \kappa_j(\varsigma_j) \leq p \wedge \sum_{i=k}^{j-1} (\kappa_i(\varsigma_i) - p) \leq q \tag{7.10}$$

Systems with storage abilities—and that can use resources faster than they replenish them—are affected by this property. An example of a recoverable system is the human muscle tissue: it can perform at maximum intensity for a short time consuming a molecule called *adenosine triphosphate*, or ATP [52], which is available in limited quantities in our body. Then, to recover, the muscle needs to decrease the intensity of the effort, and allow other metabolic pathways to replenish the initial ATP storage.

**$\langle z, r \rangle$-resilience** Having explained the concepts of resistance, functionality, and recoverability, we can finally define the resilience of SSTs as a property-aggregating property.

**Definition 4.** *Given a state trajectory $SST = (CBS_0, \varsigma_0), (CBS_1, \varsigma_1), \ldots,$ a natural number $z \in \mathbb{N}^*$, and a positive threshold $r \in \mathbb{R}^+$, $SST$ is said to be $\langle z, r \rangle$-resilient if and only if all its sub-trajectories of length $z$ are $r$-functional.*

As it was observed by Schwind *et al.* [148], using this definition, resilience is strongly interconnected with the three previous properties: by setting the parameter $z$ to 1 or $|SST|$, resilience becomes equivalent to $r$-resistance or $r$-functionality, respectively. Moreover, Schwind *et al.* [148] proved that "a finite SST is $\langle p, q \rangle$-recoverable if it is $\langle z, (p + q/z) \rangle$-resilient $\forall z \in \{1, \ldots, |SST|\}$".

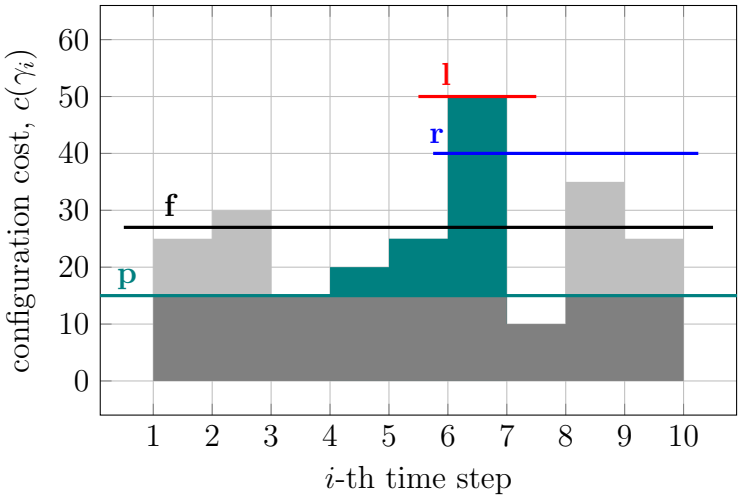



Figure 7.2 The example of a cost trajectory that is 50-resistant, 27-functional, $\langle 15, 50 \rangle$-recoverable, and $\langle 4, 40 \rangle$-resilient, according to the Definitions 1 to 4 and the Equations 7.8 to 7.10 provided in this work.

## 7.4 Complexity of Efficient Exact Inference

In the Methods section, we describe how to use the c-HMM framework to define the random variables associated to trajectories of states, observations, and costs, i.e., the probabilistic analogues of SSTs. We then show that the probability of a trajectory of costs can be derived from those of trajectories of states. However, a major assumption of our work is that only trajectories of observations are available to study the partially observable stochastic system. Because of this reason, we introduce an efficient algorithm to perform the exact inference needed to find the probability of a trajectory of states from a trajectory of observations. The

algorithm is also detailed in the Methods sections. Here, we study its complexity starting from the consideration that its last step requires the multiplication of three separate probability values (factors) that we called $\Upsilon_0$, $\Upsilon_1$, $\Upsilon_2$.

We assume that querying the sensor and the transition models for any of their elements involves a constant and negligible delay. The computation of $\Upsilon_0$ is the quickest: starting with an initialization value of 1, we need to multiply it $T$ times (the length of the time horizon of the assessment or prediction of a property) by the correct entry of the sensor model. Therefore, the time complexity of $\Upsilon_0$ is $O(T)$. The factor $\Upsilon_1$ is obtained through $|\Omega(S)|$ multiplications and $|\Omega(S)| - 1$ sums— to find $P(S_0 = s_0)$—and $T - 1$ products by entries of the transition model. Its time complexity is equal to $O(|\Omega(S)| + T)$. Finally, the algorithm in [41] has a run time of $O(|\Omega(S)|^2 \cdot T)$, plus $|\Omega(S)| - 1$ additions to compute $\Upsilon_2$. Hence, the computation of $\Upsilon_2$ using the Forward algorithm is the slowest of the three. Indeed, this is also the overall time complexity of the algorithm:

$$O(|\Omega(S)|^2 \cdot T) \tag{7.11}$$

We observe, in fact, that all the three factors $\Upsilon_0$, $\Upsilon_1$, and $\Upsilon_2$ are independent (from a computational point of view, not with regard to probability) and they can be can be easily computed in parallel, with the last one strictly dominating the others.

Most importantly, we remark that, if the time horizon is much larger than the number of states (i.e., $T \gg |\Omega(S)|$), the probabilistic inference algorithm has an overall time complexity dominated by $O(T)$. This result reveals that our algorithm—despite answering the different kind of queries we are interested in—belongs to the same time complexity class of other well-known inference algorithms for HMMs: the Forward-backward algorithm for the computation of smoothed marginals distributions, and the Viterbi algorithm for the computation of the most likely sequence of hidden variables [143].

The data structures necessary to represent the c-HMM framework have moderate memory requirements: $P(S_0)$ has size of $O(|S|)$, $P(O_t \mid S_t)$ of $O(|\Omega(S)| \cdot |\Omega(O)|)$, $P(S_{t+1} \mid S_t)$ of $O(|\Omega(S)|^2)$, and $c$ of $O(|\Omega(S)|)$. The input of our inference queries consists of two vectors, a trajectory of states $s_0, \ldots, s_T$ and a trajectory of observations $o_0, \ldots, o_T$, having size of $O(T)$ each. The computation of $\Upsilon_0$ requires to iteratively multiply the result of a previous product and store a single floating point value, hence, its space complexity is $O(1)$. Similarly, the factor $\Upsilon_1$ can be computed by repeatedly storing the result of successive additions and multiplications in the same memory cells and it has space complexity of $O(1)$. Finally, the execution of the algorithm to find $\Upsilon_2$ [41] demands memory of $O(|\Omega(S)|)$, again dominating the other two factors.

As a result, the memory requirements for the computation of the probability of a trajectory of states, given a trajectory of observations are:

$$1)\ \text{model: } O(|\Omega(S)| \cdot |\Omega(O)| + |\Omega(S)|^2); \quad 2)\ \text{input: } O(T); \quad 3)\ \text{algorithm: } O(|\Omega(S)|) \tag{7.12}$$

This also means that the space complexity of the inference algorithm itself do not depend on the time horizon $T$. In most practical cases, in which $T \gg |\Omega(S)|$, the memory bottleneck will be represented by the memories dedicated to the storage of the input sequences $s_0, \ldots, s_T$ and $o_0, \ldots, o_T$. Figure 7.3 shows how time and space complexity evolves with respect to the size of the inputs.

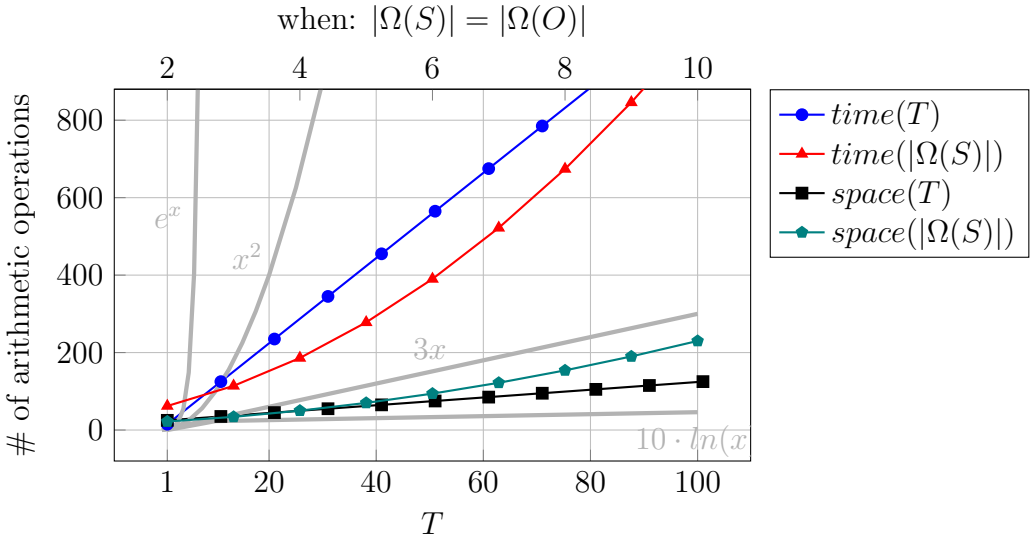



Figure 7.3 Theoretical complexity growth of the proposed inference algorithm with respect to the time horizon $T$ and the size of the state domain $|S|$. In the legend, *time* and *space* stand for time-complexity and space-complexity, respectively.

## 7.5 Complexity of Generic Property Checking

The complexity analysis in the previous section was that of an algorithm capable of computing the probability of a trajectory of states, given an assignment of the trajectory of observations. We have shown that this can be done, rather inexpensively, in time $O(T)$: a result that makes our algorithm as good as the best state-of-the-art algorithms for exact inference in HMMs. However, probabilistic property checking in c-HMMs requires an additional step: the identification of those trajectory of states that actually enforce a certain property. The complexity of this step is, in general, property-dependent.

The number of all possible assignments of a trajectory of states, is equal to $|\Omega(S)|^T$. Unless a number of impossible (i.e., zero-valued) states or transitions appear in the HMM in either $P(S_0)$ or $P(S_{t+1} \mid S_t)$, all these assignments will have non-null probabilities. However, it can be noted that properties are functions of (i.e., only depend on) trajectories of costs.

**Proposition 1.** *Given a finite time horizon $T$ and a c-HMM $\langle P(S_0), P(O_t \mid S_t), P(S_{t+1} \mid S_t), c : \Omega(S) \to \mathbb{R}^+ \rangle$, the number of possible assignments of the trajectory of costs is equal or smaller than the cardinality of the set of trajectories of states.*

*Proof.* This holds true as a consequence of the properties of the function $c$: $\forall q \in \{q \mid q = c(s) \wedge s \in \Omega(S)\},\ |c^{-1}(q)| \geq 1$. □

On the other hand, the number of trajectories of states that share the same trajectory of costs is: $\prod_{i=1}^{T} |c^{-1}(k_i)|$ (where $k_i$ is the cost of the state at the $i$-th time step). Let $K \subset \mathbb{R}^+$ be the set of all the costs that are images of the possible assignments of $S$: $K = \{k \mid k = c(s) \wedge s \in \Omega(S)\}$, the largest $max_{k \in K} |c^{-1}(k)|$, the smallest the size of the search space of the trajectories of costs were the property checking actually happens. Contrarily, if $max_{k \in K} |c^{-1}(k)| = 1$, property checking over trajectories of costs is isomorphic to property checking over the assignments of trajectories of states.

### 7.6 Bounding the Probability of l-resistance

The main challenge of dealing with long time horizons $T$ is that, as the number of possible trajectories grows exponentially with their length, the subset of trajectories that satisfy a certain (resilient or not) property might grow as well. This is also true and especially important for the resilient property of $l$-resistance.

**Proposition 2.** *This additional layer of complexity cannot really be circumvented, i.e., in general it is not possible to preclude the exponential growth of the number of probability values that must be evaluated to assess the probability of $l$-resistance.*

*Proof.* This derives from fact that $P(S_1 \leq l \wedge S_2 \leq l | o_1, o_2)$ cannot be factorized into $P(S_1 \leq l | o_1, o_2) \cdot P(S_2 \leq l | o_1, o_2)$ because $P(S_1 \leq l | o_1, o_2) \not\perp\!\!\!\perp P(S_2 \leq l | o_1, o_2)$, excluding the application of the principles of induction through an iterative algorithm. □

Instead, we can use the algorithm detailed in the Methods section to compute approximated probability values of $l$-resistance that are strictly smaller—or larger—than the actual value, i.e., plausible lower and upper bounds of $P(S_0 \leq l \wedge \cdots \wedge S_T \leq l | o_0, \ldots, o_T)$. To compute a

pessimistic estimate of this probability, we must construct new pseudo- transition and sensor models $\hat{P}(S_{t+1}|S_t), \hat{P}(O_t|S_t)$. We aggregate all the states that have cost $\leq l$ or $> l$ into two macro-states $s_{\leq l}$, $s_{>l}$ so that:

$$\begin{aligned}
\hat{P}(s_{t+1} = s_{\leq l}|s_t = s_{\leq l}) &= \min_{\substack{\text{i s.t. } c(s_i) \leq l, \\ \text{j s.t. } c(s_j) \leq l}} P(s_{t+1} = s_i|s_t = s_j) \\
\hat{P}(s_{t+1} = s_{\leq l}|s_t = s_{>l}) &= \min_{\substack{\text{i s.t. } c(s_i) \leq l, \\ \text{j s.t. } c(s_j) > l}} P(s_{t+1} = s_i|s_t = s_j)
\end{aligned} \tag{7.13}$$

$$\begin{aligned}
\hat{P}(s_{t+1} = s_{>l}|s_t = s_{\leq l}) &= \max_{\substack{\text{i s.t. } c(s_i) > l, \\ \text{j s.t. } c(s_j) \leq l}} P(s_{t+1} = s_i|s_t = s_j) \\
\hat{P}(s_{t+1} = s_{>l}|s_t = s_{>l}) &= \max_{\substack{\text{i s.t. } c(s_i) > l, \\ \text{j s.t. } c(s_j) > l}} P(s_{t+1} = s_i|s_t = s_j)
\end{aligned} \tag{7.14}$$

and, $\forall o_j \in \Omega(O)$, we have:

$$\begin{aligned}
\hat{P}(o_t = o_j|s_t = s_{\leq l}) &= \min_{i \ s.t. \ c(s_i) \leq l} P(o_t = o_j|s_t = s_i) \\
\hat{P}(o_t = o_j|s_t = s_{>l}) &= \max_{i \ s.t. \ c(s_i) > l} P(o_t = o_j|s_t = s_i)
\end{aligned} \tag{7.15}$$

Plugging this new models in our algorithm, one can compute a lower bound, i.e., a value smaller or equal, for the probability of the trajectory being $l$-resistant. Similarly, one can compute an upper bound for the probability of resistance, using the pseudo- transition and sensor models of $s_{\leq l}$ obtained by swapping the min and max operators in the definitions above. Therefore, the new model and our algorithm allow to compute, with time complexity that is linear with the length of the trajectory, an interval $[P_{low}, P_{up}]$ that certainly contains the $l$-resistance probability $P(S_0 \leq l \wedge \cdots \wedge S_T \leq l|o_0, \ldots, o_T)$.

## 7.7 Application Scenarios

To demonstrate the potential of the proposed methodology, we apply both its modelling and probabilistic inference facets to four practical scenarios. These examples serve to demonstrate that resilience and the resilient properties have a prominent role in several different domains. Moreover, they show that, in the "random world" [67] we frequently encounter environments that have non-deterministic dynamics and are observed through noisy, imperfect, or broken sensors (i.e., partial observability). The first two qualitative examples are inspired by the domains of disaster management and macroeconomics. The third and fourth example are drawn from the fields of self-adaptive computing for aerospace applications and swarm

robotics, respectively, and they are used to evaluate the quantitative aspects of the proposed approach as well.

### 7.7.1 Disaster Management

When dangerous disruptive events occur, proper disaster management is crucial to protect human lives and minimize casualties [124]. Effective disaster management cannot be decoupled from good modelling and decision making strategies [102]. In our first application scenario, we model a four islands archipelago $X_0$, $X_1$, $X_2$, $X_3$ (see Figure 7.4) that can be affected by three different level of alert $\mathcal{D}(X_i) = a_0, a_1, a_2$—from "no intervention needed" ($a_0$) to "emergency" ($a_2$), passing by "some intervention needed" ($a_1$). In this example, $|\Omega(S)| = 3^4 = 81$. To take into account the different speeds at which alerts escalate and get re-absorbed on each islands, we define four transition models $\forall i \in [0, 3]$, $P_i(\{X_i\}_{t+1}|\{X_i\}_t)$ and construct the overall probabilistic dynamics as $P(S_{t+1}|S_t) = P(\{X_0, X_1, X_2, X_3\}_{t+1}|\{X_0, X_1, X_2, X_3\}_t) = \prod_i P_i(X_{t+1}|X_t)$ (this also implies that the alert status as independent from one another). We assume that the observations domain is isomorphic to that of the states $\Omega(O) = \Omega(S)$, that an "emergency control centre" resides on the $j$-th island, and that the reliability of an observation decays exponentially with the distance it has to travel (this is, for example, the case of a multi-hop communication network with constant packet drop between any two nodes but one could also choose to plug-in any of the more sophisticated probabilistic models found in the literature [177]). Having defined the observation of the $i$-th island status as $O^{X_i}$ and its distance from the control centre as $d_i$, then $P_i(O_t^{X_i} = a_p|X_t = a_q) = e^{-d_i}$ if $a_p = a_q$, and $(1 - e^{-d_i})/(|\Omega(O)| - 1)$ otherwise. The overall observation model is defined as $P(O_t|S_t) = P(\{O^{X_0}, O^{X_1}, O^{X_2}, O^{X_3}\}_t|\{X_0, X_1, X_2, X_3\}_t) = \prod_i P_i(O_t^{X_i}|X_t)$. Functions $c_i(X_i) : \mathcal{D}(X_i) \to \mathbb{N}$ state how many resources, e.g., the number of search and rescue teams, have to be sent to the $i$-th island, depending on its alert status. The cost function $c(S) : \Omega(S) \to \mathbb{N}^4 = \sum_i c_i$ reveals how many resources are required to cope with each situation in the state domain.

The numerical values in the transition and the sensor model can be found or improved upon used historical data and expert knowledge. The emergency control centre can use them to look at the stream of information about the alert status (or their prediction through the transition model $P(S_{t+1}|S_t)$). Performing inference on the system model allows to answer different queries of interest. If a limited number of search and rescue teams are present on the archipelago, computing the probability of the $l$-resistance property to hold true and making sure that it is above a desirable threshold (e.g., $p(\phi(resistance, l)) \geq 0.95$), ensures that $l$ is the correct number of resources to deal with the potential emergencies—should the resistance

probability drop, more resources would be necessary. Furthermore, if the archipelago can temporarily recall an additional $q$ resources (e.g., from a national guard), the $p$ parameter for which the probability of $\langle p, q + l\rangle$-recoverability is above a safe limit will tell for how many time steps (days or months) those extra resources should be mobilized (and therefore paid, quartered, etc.).

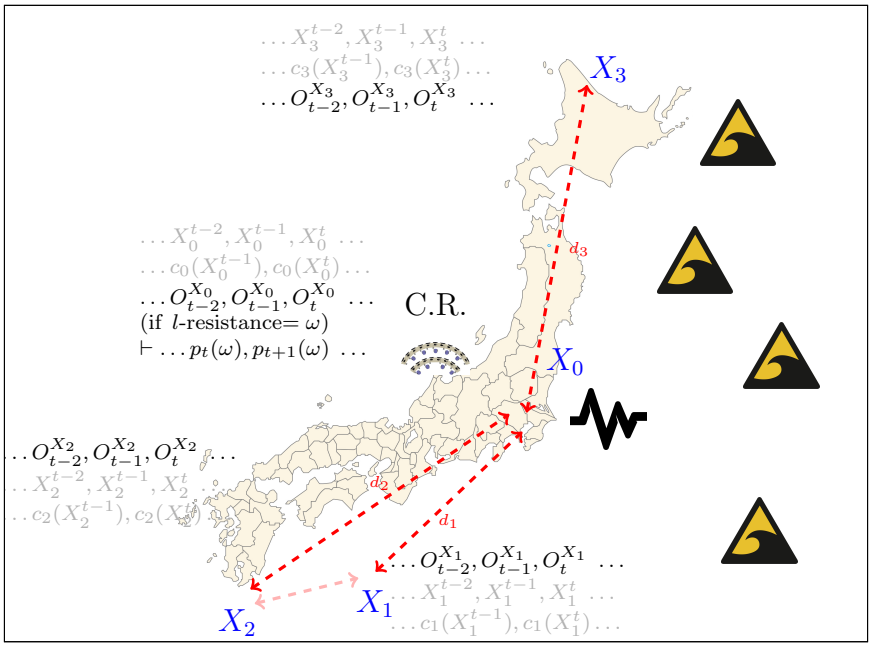



Figure 7.4 The four island archipelago modelled in the first application scenario. For each island, the image shows its geographical distribution, the evolving state, cost, and (partial) observation from the point of view of the control room. (Figure created using TikZ/PGF v3.0.0, GIMP 2.8.14, and cliparts from `http://openclipart.org`.)

### 7.7.2 Macroeconomics

Probabilistic and statistical models are already widely exploited tools in the fields in economics and finance—the latter especially. A notable example being the research on the expected return and risk of efficient portfolios by Harry Markowitz [97]. The deterministic modelling approach of traditional macroeconomics, on the other hand, has come to be questioned over the last decade by the crisis of 2008 and the growing prominence of experimental and behavioural economics [44]. Reckoning the existence of yet many unknowns in modern macroeconomics, probability theory only seems the natural development direction for models that need to be able to account for the uncertainties of this domain and the irrationality of human behaviours.

Applying the proposed modelling to the context of macroeconomics gives us a tool to derive valuable insights this world. Assessing the resilience of a macroeconomic system is important for multiple reasons: smoothly running economics guarantee the development, stability, and fairness of our societies. The 2008 housing market crisis proved that existing models are not enough to protect us from rare, non-directly observable, and counterintuitively correlated events [88]. The argument that macroeconomics should be revisited to deal with the uncertainty of the real world is not new [154] and statistical model for certain phenomena have been proposed [38]. Existing research can be leveraged by our approach by simply verifying that the Markov property is enforced $P(S_{t+1}|S_t)$. The general consensus on the yet incomplete understanding of macroeconomics lends itself perfectly to a partially-observable modelling approach $P(O_t|S_t)$. As economists are well aware of the limitations of existing models, they often rely on stress tests [45] to evaluate the resilience of financial institutions [50]. Stress tests are experimental tools that go beyond statistical analysis but, for which, statistical meta-analyses exist [127] and can be used to construct the observation model required by our approach. Intuitively, the cost function of the c-HMM describing this scenario will tell which amount of money (in cash, deposits, or bonds) a government would need to prevent a default in a certain state. Governments, banks, and investment funds typically monitor time horizons of 5, 10 (sometimes 15, 20) years. In this context, the $f$-functionality property represents the amount of funding that has be made available, on average, across multiple year budgets. The $\langle p, q\rangle$-recoverability property tells how far into debt a government would have to go to recover from a crisis within a fixed timeframe.

### 7.7.3 Self-adaptive Computing

Self-adaptive computers possess *ad hoc* capabilities—e.g., sensors, actuators, and decision making loops [120]—that allow them to express autonomous behaviours. Because they do not require the supervision of a human operator, these systems are especially suitable for critical, advanced applications such as space systems and robotic exploration. An autonomous computer and a resilient ecological system share several properties, for example the ability to self-protect and self-heal [84] and assessing the resilience of the first is of primary importance both at design and run time. Previous research [115, 117, 95] proved that probabilistic models have the potential to enable autonomous computing systems. We now demonstrate how they can be exploited for the analysis of their resilience.

The ArduSat Payload Processor Module (ASPPM) carried by the 1U CubeSat[170] ArduSat-1 consists of one supervisor processor and 16 processing elements (PEs), and it is the ideal platform for a modular, redundant autonomous on-board computer (OBC). The resilience of

the OBC of a spacecraft is typically enforced through the software and/or hardware replication of its essential functionalities: (1) housekeeping (C&DH), i.e., all the software tasks contributing to the monitoring of the satellites status and the correct execution of its routine functions; (2) the processing of the data collected by the payload of the satellite while performing its mission (Mission), e.g., running a classification algorithm over the images captured by a camera [46]; and (3) the attitude control algorithm (ADCS), responsible for the proper orientation of the satellite with respect to Earth and its targets, through the computation of the control signals of the satellite actuators (e.g., reaction wheels).

Because of the harsh toll posed by space weather (solar wind, cosmic rays) on electronics, each of the processing elements $\forall i \in [0, |PE| - 1]$, $pe^i$ in the set $PE$ can find itself in one of three states: $pe^i \in \{w, t, p\}$, that is, correct operation $w$, experiencing a transient fault state $t$, or permanent failure $p$. A stochastic transition model describes the ageing of a PE [115] and it is parameterized by the impact rate of particle radiation $r$ and the mean time to failure $MTTF$ of a PE. These parameters are responsible for transient and permanent faults, respectively. $r$ is strongly orbit-dependent and is computed with the aid of radiation models such as Creme96 and SPENVIS [164, 65]. Assuming independence among the evolution of the PEs and defining the state of the system as $S = \{pe^i \ s.t. \ i \in [0, |PE| - 1]\}$, we can generalize the transition model [115] as follows (where $W$ is the duration of a time step):

$$P(S_{t+1}|S_t) = \prod_{i=0}^{|PE|-1} P(pe_{t+1}|pe_t) = \\ = \begin{cases} \text{if } pe_t^i = (w||t)\text{: } P(pe_{t+1}^i) = \langle \frac{1-r}{e^{W/MTTF}}, \frac{r}{e^{W/MTTF}}, 1 - \frac{1}{e^{W/MTTF}} \rangle \\ \text{if } pe_t^i = p\text{: } P(pe_{t+1}^i) = \langle 0, 0, 1 \rangle \end{cases} \tag{7.16}$$

In the case of ArduSat-1, the observers of the resilient system are the ASPPM's on board supervisor ATmega2561 microcontroller and the external NanoMind A712C flight control computer. Observations of each PE, however, are not perfect for two reasons: (1) errors can slip into the observers too; and (2) transient and permanent faults are, *a priori*, indistinguishable. Our approach seamlessly models these kinds of observations with a framework that accounts for both "partial" (in modal logic, $\neg\Box(\Omega(S) = \Omega(O))$) and probabilistic observability. Having defined the observation of each PE as working or faulty, $O^{pe_i} \in \{w, f\}$, and the system observation as the set of observation of all PEs, $O = \{O^{pe^i} \ s.t. \ i \in [0, |PE| - 1]\}$, we can use any suitable memoryless probability distributions for the sensor model [115] (with

false positive and false negative rates of $p^{fp}, p^{fn}$):

$$P(O_t|S_t) = \prod_{i=0}^{|PE|-1} P(O_t^{pe^i}|pe_t^i) = \begin{cases} \text{if } pe_t^i = w\text{: } P(O_t^{pe^i}) = \langle 1-p^{fp}, p^{fp}\rangle \\ \text{if } pe_t^i = (t||p)\text{: } P(O_t^{pe^i}) = \langle p^{fn}, 1-p^{fn}\rangle \end{cases} \tag{7.17}$$

The cost function expresses the utility [143] of a configuration, that is, the scientific data throughput (e.g., in MBytes per orbit or per day) that a certain state configuration puts on the downlink of the satellite's telecommunication system. In general, this data throughput is a function of the state of the ASPPM $s \in S$, the orbit of the satellite $\xi \in \Xi$, and number/position of ground stations $\psi \in \Psi$: $ST(s, \xi, \psi) : S \times \Xi \times \Psi \rightarrow \mathbb{R}^+$. For a given low-Earth orbit $\bar{\xi}$ with a 400km altitude and 51° inclination, and a single ground station $\bar{\psi}$ in North America, we write $c^{\bar{\xi},\bar{\psi}}(S)$ as the cost function of the ASPPM state as:

$$c^{\bar{\xi},\bar{\psi}}(S) = ST(S, \bar{\xi}, \bar{\psi}) =$$
$$= \begin{cases} \dots \\ 3.7MB/day \text{ if } S^{i-1} = \langle \varnothing; \varnothing\rangle;\ map:\ \langle c\,\&\,dh \mapsto pe^{9,13:14}; mission \mapsto pe^{2:4,6:8}; acds \mapsto pe^{10:12,15:16}\rangle \\ 2.0MB/day \text{ if } S^{i} = \langle \varnothing; 4\rangle;\ map:\ \langle c\,\&\,dh \mapsto pe^{9,13:14}; mission \mapsto pe^{2:3,6:7}; acds \mapsto pe^{10:12,15:16}\rangle \\ 1.3MB/day \text{ if } S^{i+1} = \langle \varnothing; 4, 12\rangle;\ map:\ \langle c\,\&\,dh \mapsto pe^{9,13}; mission \mapsto pe^{3,7}; acds \mapsto pe^{10:11,15}\rangle \\ \dots \end{cases} \tag{7.18}$$

In Equation 7.18, the shortcut $\langle \varnothing; 4\rangle$ is used to indicate a state in which no PE is experiencing a transient fault, and $pe^4$ is permanently faulty (all other PEs are assumed to work correctly); *map* specifies how the software tasks are mapped to the PEs in any given state. Figure 7.5 offers a visual reference of the subset of these mappings, as in Equation 7.18.

To discover meaningful semantics associated to the resilient properties, we introduce a helper (cost) function $\hat{c}(S) = \bar{c} - c(S)$, where $\bar{c}$ is the theoretical maximum throughput attainable by the satellite. Using $\hat{c}(S)$, computing the probability distributions of $f$-functionality and $l$-resistance reveals the expected and worst-case data throughput sent to Earth, respectively. The property of $\langle p, q\rangle$-recoverability can help quantify the loss of scientific data in the case of drops in the throughput (due to faults or reconfiguration of the system). The advantage of using the algorithm proposed in this work (see Methods) to assess these properties is the ability to maintain the computation within reasonable time limits, even for relatively long

traces and complex models. In a search space of $3^{16}$ states, $2^{16}$ possible observations, and time horizons of 10, 100, or 1000 steps, the proposed approach requires a number of arithmetic operations in the order of $10^{13-15}$ to compute the probability of a state trajectory. The same problem would simply be intractable by any other algorithm that requires to evaluate the $10^{75}$ entries in the conditional joint probability distribution (CJPD). Because of the exponential growth of the CJPD, the savings are remarkable (order of $10^{59}$) even for properties that are satisfied by a large (e.g., 30%) fraction of the possible state trajectories.

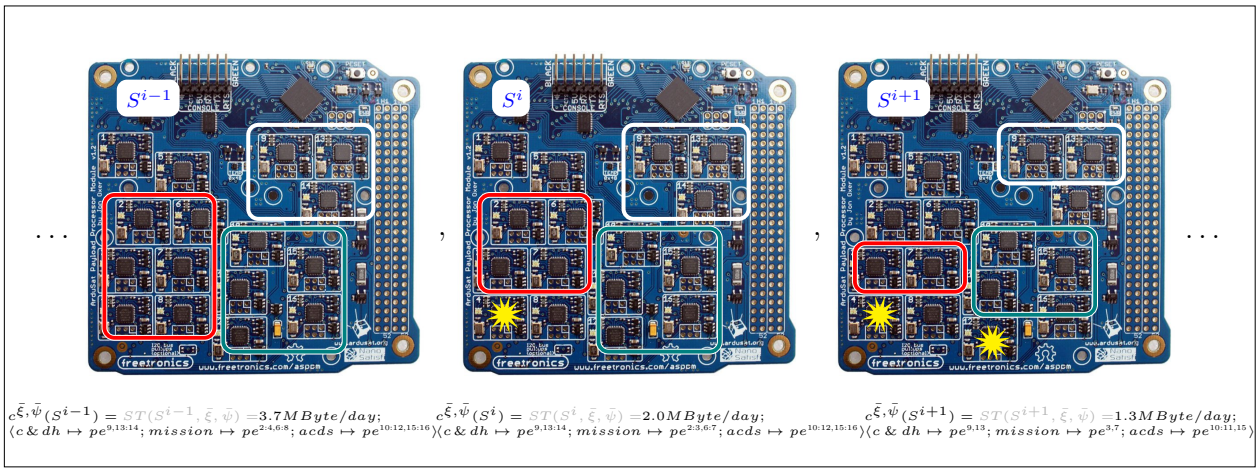



Figure 7.5 A visual representation of three of the possible "software task"-to-"hardware resource" mappings in the state space of the 1U CubeSat's Arduino-based ASPPM from the third application scenario, as presented in Equation 7.18.

The potential of self-aware computing is not limited to satellites. Studying the challenges of Mars rover operations, Gaines *et al.* [51] outlined a model of seven factors impacting productivity. Among non-human factors, they identified the reliability of the uplink/downlink as a cause for "deferred" sols—i.e., Martian solar days in which the campaign objectives have to be postponed to address unexpected issues. Indeed, they suggest "state-aware health assessment" as one of the capabilities that shall be developed in future missions to mitigate this problem.

For example, NASA and JPL's most recent Mars rover, Curiosity, is able to perform ∼5h/sol of tactical science activities [58]. This is due to the fact that direct-to-Earth communication is limited—by power and orbital constraints—to a few hours/day at data rates of 0.5 to 32kb/s. Therefore, most transmissions are relayed by two sun-synchronous orbiters—Mars Reconnaissance Orbiter, at up to 2Mb/s, and Odyssey, at 128 or 256kb/s. Each of the orbiters passes over the rover, every sol, for a 8'-window while they can both transmit to Earth for ∼16h/day. Commands are uploaded to the rover every sol during an overnight orbiter pass

(or direct-from-Earth at local midmorning). Data that are necessary to plan the activities of the following sol are then returned via an orbiter telecom pass in the midafternoon. Non-essential information is stored and returned during the following overnight orbiter pass [58]. As a consequence, if the rover fails to send the required information during the correct orbiter pass, the tactical team might not be able to plan the activities for the following sol. This is an issue that will aggravate in the near future, as the current fleet of sun-synchronous orbiters is replaced with non-sun-synchronous orbiters [51].

As originally planned, Curiosity's primary mission spanned over 669 sols. Accounting for (i) a commissioning phase of 90 sols, (ii) 30 sols of solar conjunction, (iii) 10 sols for maintenance and updates, (iv) a 20% of "not commandable" sols due to Earth-Mars phasing, and (v) a 25% of "non-productive" sols "due to unforeseen shortfalls in mission resources [...] or communication problems" [58], the rover was left with ∼300 sols to explore the vicinity of the Gale crater, traverse ∼18km, and collect ∼11 samples. With hindsight, the 25% estimate of "non-productive" sols proved to be rather conservative: the study in [51] observes that tactical activities were only deferred in 3 out of 19 (16%) sols during 2014's Pahrump Hills campaign and in 1 out of 24 (4%) sols during 2015's Artist's Drive. Yet, self-aware computing might have the potential to further improve performance, e.g., with the implementation of a decision support system (DSS) on top of the self-assessment framework described in this work.

Having associated probability values to the data throughput of a computing system (through a model as the one in Equations 7.16 to 7.18), a binary classification/decision system would autonomously choose whether to use the overnight orbiter pass to (i) transmit the non-essential information (the default behaviour) or (ii) re-transmit the data required for tactical planning (when it believes that the previous transmission failed) and prevent unproductive sols. The sensitivity and specificity of the classifier are affected by several factors (including the noisiness of the on-board sensors and the time horizon of the assessment algorithm). However, even assuming relatively weak performance (e.g., sensitivity and specificity of 0.8) and the conservative "deferred sol" incidence of [51], this DSS could reduce the number of unproductive sols by 3.2–12.8%. Over the course of the >1600 sols spent by Curiosity on Mars, it means 50-to-200 extra sols of science activities, equivalent to 3-to-12 extra kilometres and 2-to-7 additional samples.

### 7.7.4 Swarm Robotics

As many-robot systems, or robot swarms, become more and more pervasive, researchers must devise new, efficient ways to control and coordinate them [22]. In the fourth practical scenario,

we test our framework in the context of the networked multi-robot system of Figure 7.6, where robots move independently and have a limited communication range. We implemented a simulator for the robots' movement and communication model, the proposed algorithm, and an alternative reference approach based on the computation of the conditional joint probability distribution. We remark that computing the CJPD is already a more efficient approach than blindly expanding the entire joint probability distribution of a c-HMM. We analyze the scenario with two examples: a small one with 4 robots in a 20cm by 20cm arena and a large example with 20 robots in a 40cm by 40cm area. In both examples, robots have a diameter of 2cm (similarly to Kilobots [142]), move on independent random walks at a speed of 2cm/s, and have a communication range of 10cm.

As a transition model, we use the conditional probability distribution that describes the way in which the number of neighbors $R$ of a robot evolves over a time step of 1s: $P(R_{t+1}|R_t)$. To empirically derive this model, in both examples, we performed 30 random-walk simulations of $10'$ each, with the positions of the robots randomly initialized. For the sensor model, we assume that communication links between neighbors can be temporarily broken with probability $d = 0.1$. As a consequence, the sensor model that describes the number of robots $V$ that are actually visible to a robot with $R$ neighbors follows the binomial distribution: $P(V_t|R_t) = P(X = V_t)$ with $X \sim B(R_t, d)$. We want to assess the probability of the following two properties:

— $\Lambda$ ($l$-resistance), given a series of observations over a time horizon $T$ varying from 4 to 6, property $\Lambda$ guarantees that a robot always maintained more than $l$ neighbors. The $l$ parameter is set to 2 in the small example and 10 in the large one.

— $\Theta$: given a series of observations over a time horizon $T$ varying from 4 to 6, property $\Theta$ says that a robot lost connectivity (i.e., found itself in a position with zero neighbors) precisely during the last timestep—and not before.

Table 7.1 reports the results of specific experiments, taking typical series of observations as inputs. It is worth noting that, because the proposed one is an exact approach, the obtained probability values are identical w.r.t. those extracted from the CJPD, while—from the results of the experiments—it emerges that the computational time is reduced by a factor ranging between $10^2$ and $10^4$. In the 4 robots/6 steps time horizon case, for example, the computational time of the probability of property $\Lambda$ is lowered from >1000s to ∼0.01-0.1s.

Figure 7.7 compares the time delay of the proposed approach and that required by the computation of the CJPD. For large scenarios, the CJPD delay rapidly gets off the chart. The proposed algorithm, instead, allows to deal with 20 robots with a comparable, but smaller, delay than the one required by the computation of the CJPD in the 4 robots scenario. In

particular, we observe that the advantage of the proposed approach over the use of the CJPD actually increases with the length of the time horizon and the number of robots in the scenario. Being able to perform exact inference in only seconds in large scenarios, accounting for tens of robots, this approach can effectively be implemented in several practical multi-robot applications, such as target tracking, area coverage, or task allocation. Unlike previous work on resilient robot formations and partially-observable robot swarms [146], the proposed approach does not limit the movement of the robots into configurations whose resilience can be established *a priori* but rather it allows the *a posteriori* assessment of resilience in a distributed fashion.

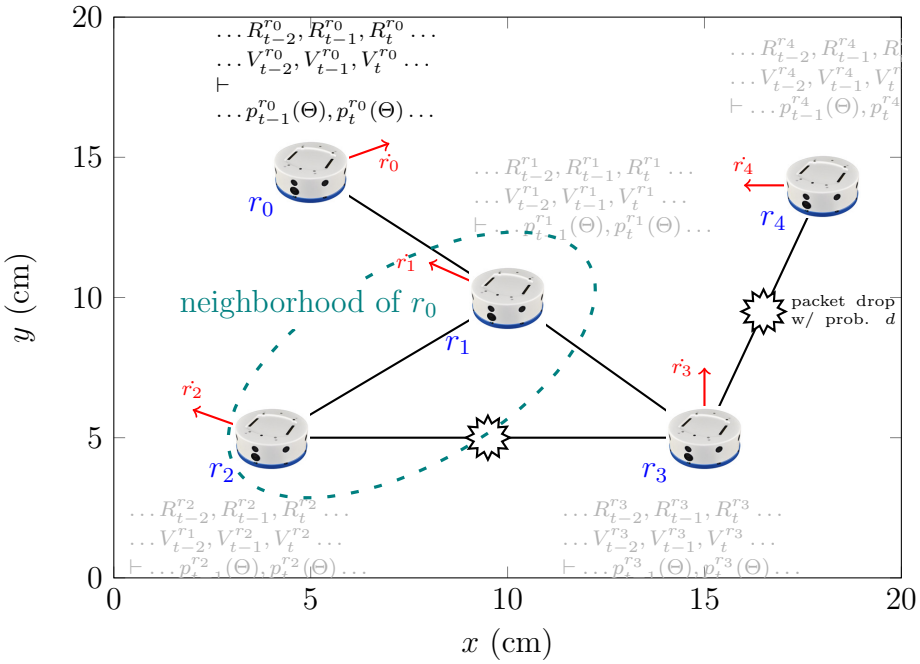



Figure 7.6 A robotic swarm, as described in the fourth application scenario. Each robot possesses a position, velocity, state (the number of its neighbors), and a partial observation (of its neighborhood) evolving over time. The inference algorithm is executed locally to assess the probability of losing connectivity with respect to the rest of the swarm at each time step.

## 7.8 Discussion

Summing up our work, we adapted the CBS/DS-based formalization of resilience given by Schwind *et al.*[148] (composed of the resilient properties of resistance, functionality, and recoverability) to the timed probabilistic framework of hidden Markov models. To do so, we defined the extended framework of c-HMMs. In the Methods section, we outline a state-of-the-art inference algorithm able to answer the queries required for the probabilistic property

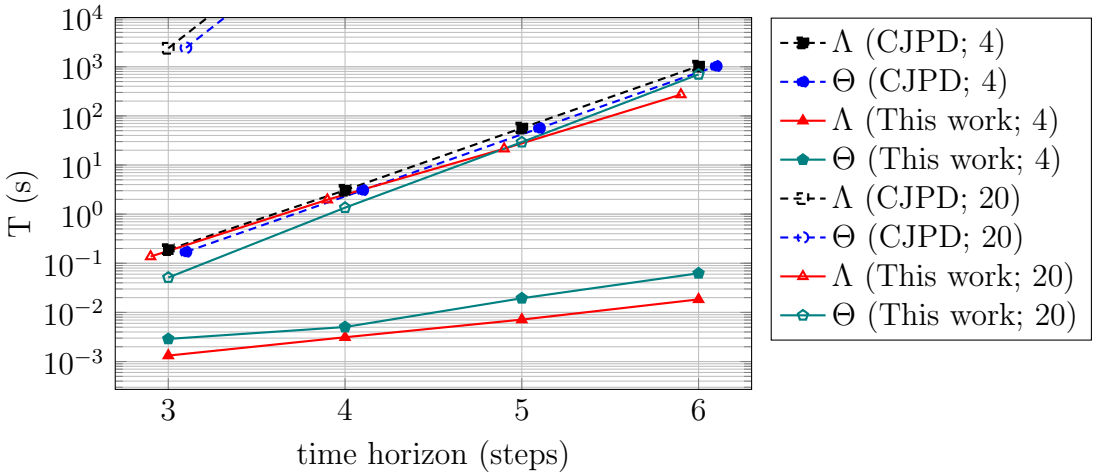



Figure 7.7 Experimental assessment of the time complexity and comparison of the scalability of the computational time of different queries for property $\Lambda$ and property $\Theta$ through the algorithm proposed in this work versus expanding the conditional join probability distribution, in the 4 robots and 20 robots scenarios.

checking of resilience over this model. Furthermore, we studied the space- and time- complexity of this inference algorithm as well as those of property checking.

We demonstrated the practical applicability of our approach in four qualitative and quantitative scenarios of growing technical complexity. In our experimental evaluation, we implemented the algorithm in the Matlab-compatible scripting language GNU Octave (see the Additional Information for the supplementary materials) and tested it in the autonomous multi-processor computing system of a nano-satellite and in a multi-robot scenario to answer queries about the robots' connectivity. The experimental results show that, even in small domains, the proposed approach is approximately (1) four orders of magnitude faster than expanding the full conditional joint probability distribution. Furthermore, the scenarios revealed that the proposed approach is capable of (2) modelling partial observability in a way that deterministic models cannot grasp and (3) leading to insights about resilience that would be, otherwise, concealed—e.g., the link between the extra resources required to probabilistically ensure $\langle p, q \rangle$-recoverability and the tightness of the associated deadline.

Looking forward, the opportunities for the further development of this work reside in the possible extensions of both its framework and the inference methodology. To improve the general applicability of our approach, the next step is an inference algorithm capable of dealing with missing data in the trajectory of observations. Moreover, the c-HMM framework has the potential to be enriched with the ability to perform learning, decision making, and planning—insights can be drawn from the existing frameworks of machine learning, decision

| Observation Trajectory $TO$ | # of Robots | $p$ of $\Lambda$ | Computation time (s) CJPD | Proposed |
|---|---|---|---|---|
| $[2, 2, 1, 2]$ | 4 | 0.77167 | 3.022 | 0.003 |
| $[10, 10, 8, 10]$ | 20 | 0.40485 | n/a | 1.964 |
| $[2, 2, 2, 2, 1, 2]$ | 4 | 0.77033 | 1027.0 | 0.018 |
| $[10, 10, 10, 10, 8, 10]$ | 20 | 0.38312 | n/a | 271.51 |
| Observation Trajectory $TO$ | # of Robots | $p$ of $\Theta$ | Computation time (s) CJPD | Proposed |
| $[2, 2, 1, 0]$ | 4 | 0.52712 | 3.059 | 0.005 |
| $[2, 2, 1, 0]$ | 20 | 0.54640 | n/a | 1.354 |
| $[2, 2, 2, 2, 1, 0]$ | 4 | 0.52727 | 1025.5 | 0.063 |
| $[2, 2, 2, 2, 1, 0]$ | 20 | 0.53958 | n/a | 698.14 |

Table 7.1 Experimental results from the fourth application scenario, describing a robot in the small or large swarm trying to assess the probability of properties $\Lambda$ and $\Theta$ using only local and the—possibly faulty—observations of its neighborhood.

networks, and MDPs.

### 7.9 Methods

In the section on resilience, resilient properties, and probabilistic models we have explored the concepts of $l$-resistance, $f$-functionality, $\langle p, q\rangle$-recoverability, $\langle z, r\rangle$-resilience, and hidden Markov models. We explained that the traditional HMM framework requires to become a c-HMM in order to support the formal definition of resilience given by Schwind *et al.* [149]:

$$c\text{-}HMM = \langle P(S_0), P(O_t \mid S_t), P(S_{t+1} \mid S_t), c : \Omega(S) \rightarrow \mathbb{R}^+ \rangle \tag{7.19}$$

c-HMMs extend the HMM framework with a static cost function $c$, defined over the domain $\Omega$ of its (random) state variable $S$, and taking positive values in $\mathbb{R}$. We use the single discrete state variable $S$ of a HMM as a way to represent, by enumeration, the state configuration. This information, in the constraint-based systems $CBSs$, was encoded using the set of variables $\mathbf{X}$, and assignment $\varsigma$. On top of the c-HMM framework, we can define the random variables associated to trajectory of states, $TS$, and trajectory of observations, $TO$, and show how to compute their (conditioned and unconditioned) probability distributions. All the concepts presented in this section are implemented and evaluated using the open-source Matlab-compatible GNU Octave language.

### 7.9.1 States, Observations, Costs, and Trajectories

In the SR-model, the definition of the resilient properties was based on the concepts of SSTs and their corresponding sequences of costs. In c-HMMs, we have similar constructs for states and costs, as well as observations. The main difference is that these concepts are now built on top of random variables [140] and, therefore, they also can be associated with probability distributions. Given a c-HMM and a finite time horizon $T$, we define its trajectory of states $TS$ as the sequence of state variables $S_i \ \ \forall i \in \{1, \ldots, T\}$. This can be rewritten as: $TS := S_0, S_1, \ldots, S_T$.

$S$ is a random variable and, therefore, $TS$ is also a random variable. The number of possible assignments of $TS$ grows exponentially with the time horizon: $|\Omega(TS)| = |\Omega(S)|^T$. Because the mapping provided by the cost function $c$ is purely deterministic, each assignment of $TS$ is unambiguously associated with a trajectory of costs $tc = c(s_0), c(s_1), \ldots, c(s_T)$ and $TC$ is a random variable with $|\Omega(TC)| \leq |\Omega(TS)| = |\Omega(S)|^T$.

Similar considerations are also valid for trajectories of observations $TO := O_0, O_1, \ldots, O_T$ and their possible assignments $to := o_0, o_1, \ldots, o_T$. If neither the transition model nor the sensor model contain probability values of 0, all possible sequences of states can potentially occur and produce any one of the sequences of observations. Therefore, the number of all possible configurations—i.e., entries in the joint probability distribution (JPD)—of a c-HMM is:

$$(|\Omega(S)| \cdot |\Omega(O)|)^T \tag{7.20}$$

This number, in principle, represents the maximum (worst-case) complexity of performing inference on our model. However, as we explained in the section on complexity, when dealing with real-world environments and the resilient properties, we are only interested in a very specific type of inference queries. That is, those queries that can return one of the $|S|^T$ probability values of the conditional probability distribution of $TS$ with respect to a given $to$:

$$P(TS \mid to) = P(S_0, S_1, \ldots, S_T \mid o_0, o_1, \ldots, o_T) \tag{7.21}$$

This is due to the fact that: (1) the state variable $S$ is also called the "hidden" variable as it is, in practice, never directly observable (making the actual $ts$ taken by $TS$ unknown); and (2) the values taken by the observation variable $O$ are, in most cases, the one piece of *partial/imperfect* information that we can always access.

### 7.9.2 From the Probability of Cost Trajectories to the Probability of Properties

We have seen that the resilient properties—once their parameters are fixed—can be considered as boolean attributes of sequences of costs associated to SSTs. In the context of the c-HMM framework, we say that an assignment of the trajectory of states $ts = s_0, s_1, \dots$ enforces the property $\phi$ if and only if its corresponding trajectory of costs $tc = c(s_0), c(s_1), \dots$ is satisfies the definition of that property, i.e., $\phi(c(s_0), c(s_1), \dots) = true$. Computing the probability distribution of parametric properties $\phi(k)$, such as resistance and functionality, with respect to their parameters, can provide valuable insights, as shown in Figure 7.8: a rapid drop in the probability distribution might suggest the existence of a threshold cost that is unlikely to be overcome. The probability of a property $P(\phi)$ is equal to the sum of the probabilities of all the distinct trajectories of costs in which $\phi$ holds:

$$P(\phi) = \sum_{\forall i \in \{i | \phi(tc_i) = true\}} P(tc_i) \tag{7.22}$$

In turn, the probability of a fixed assignment of the trajectory of costs $tc$ is equal to the sum of the probabilities of all the distinct trajectories of states that are mapped to $tc$ by the cost function $c$. To simplify the notation, we will also use $C(ts)$ to indicate the trajectory $tc = c(s_0), c(s_1), \dots$ resulting from the application of the cost function $c$ to the assignment $ts$.

$$P(tc) = \sum_{\forall i \in \{i | C(ts_i) = tc\}} P(ts_i) \tag{7.23}$$

Plugging Equation 7.23 into Equation 7.22, one can compute the probability of a property $\phi$ as a function of the probabilities of distinct assignments of the trajectory of states (Equation 7.24). We observe that all the possible assignments of $TS$ are "distinct" by definition, even if many of them could be mapped by $c$ to identical trajectories of costs.

$$P(\phi) = \sum_{\forall i \in \{i | \phi(tc_i) = true\}} \sum_{\forall j \in \{j | C(ts_j) = tc_i\}} P(ts_j) \tag{7.24}$$

Having assumed to be able to observe the system by its trajectory of observations $TO$, we are then interested in computing the conditional distribution of $\phi$ with respect to the assignment $to$ of $TO$. To do so, we start from Equation 7.24 and we re-write it with the addition of conditioning on both sides by $to$:

$$P(\phi \mid to) = \sum_{\forall i \in \{i | \phi(tc_i) = true\}} \sum_{\forall j \in \{j | C(ts_j) = tc_i\}} P(ts_j \mid to) \tag{7.25}$$

Equation 7.25 shows that computing the probability of a property $\phi$, given an assignment of the trajectory of observations *to*, consists of two different subproblems: (1) identifying the assignments of the trajectory of states $TS$ that map to assignments of the trajectory of costs $TC$ that satisfy the property; (2) computing the conditional probability of these assignments of $TS$ with respect to the assignment of the trajectory of observations *to*. As we discussed in the section on general property checking, the first problem strictly depends on the nature of the property we are evaluating. The second problem, instead, can be efficiently tackled in its general form by combining different inference methods for HMMs, as shown in the next section.

**Traditional HMM Inference**

The most common algorithms for exact inference in HMMs are: the forward algorithm, the forward-backward algorithm and the Viterbi algorithm [144]. The first two answer queries about marginal probabilities while the third enables maximum *a posteriori* (MAP) inference [86]. More specifically, the forward algorithm can be used to compute the probability distribution of the current (or the upcoming) hidden variable, given a sequence of observations. This operation is often referred to as filtering (or prediction):

$$\begin{aligned} P(S_T \mid o_1, \ldots, o_T) \\ P(S_{T+1} \mid o_1, \ldots, o_T) \end{aligned} \tag{7.26}$$

The forward-backward algorithm allows to refine the estimate (smoothing) of the probability distribution of a hidden variable using subsequently collected information, that is, computing:

$$P(S_N \mid o_1, \ldots, o_T) \tag{7.27}$$

when $1 \leq N \leq T$. Finally, the Viterbi algorithm allows to discover the most likely sequence of assignments of the hidden variable for a given sequence of observations:

$$\underset{s_0,\ldots,s_T}{\operatorname{argmax}} P(s_0, \ldots, s_T \mid o_1, \ldots, o_T) \tag{7.28}$$

All these algorithms have time-complexity that is linear with the length of the sequence of observations they take as input: $O(T)$ [144]. However, none of these algorithms directly provides an answer to the family of queries that we are interest in for the scope of this work. That is, the *a posteriori* probabilities of arbitrary sequences of assignments of the hidden

variable for a given sequence of observations, such as:

$$P(s_0, \ldots, s_T \mid o_1, \ldots, o_T) = P(ts|to) \tag{7.29}$$

(Note that the output of the Viterbi algorithm is only one, specific sequence of assignments and not a probability value.)

### 7.9.3 Efficient Inference

The easiest, but highly inefficient, way to find the probability value of $P(ts|to)$ consists of computing the complete joint probability distribution of the c-HMM over the time horizon $T$. Because HMMs are Bayesian networks, their JPD is equal to the chain product of all the conditional probability distributions (CPDs) in their nodes. Then, one can condition by the evidence of $to$, and finally re-normalize the entire distribution so that it sums up to 1. This is clearly an unsustainably expensive approach because computing the JPD requires time and space complexity of $O(\Omega(S) \cdot \Omega(O))^T$.

Instead, we propose a much more efficient algorithm to compute the conditional probability of a finite state trajectory assignment $s_0, \ldots, s_T$ with respect to the observation trajectory assignment $o_0, \ldots, o_T$. To do so, we start by re-writing the probability value of interest using the Bayes' theorem:

$$P(s_0, \ldots, s_T \mid o_1, \ldots, o_T) = [P(o_0, \ldots, o_T \mid s_0, \ldots, s_T) \cdot P(s_0, \ldots, s_T)] \cdot P(o_0, \ldots, o_T)^{-1} = \Upsilon_0 \cdot \Upsilon_1 \cdot \Upsilon_2 \tag{7.30}$$

Equation 7.30 shows how to decompose the problem into the computation of three factors that we call $\Upsilon_0$, $\Upsilon_1$, and $\Upsilon_2$ and can be tackled separately. Given an assignment of the state variables, the conditional probability of a sequence of observations $\Upsilon_0$ can be computed as the product of the appropriate entries of the sensor model.

$$\Upsilon_0 = P(o_0, \ldots, o_T \mid s_0, \ldots, s_T) = \prod_{i=1}^{T} P(O_t = o_i \mid S_t = s_i) \tag{7.31}$$

The probability of an assignment of the trajectory of states $TS$ (notwithstanding the values taken by the observations variables) $\Upsilon_1$, only depends on the transition model $P(S_{t+1} \mid S_t)$ and the ground belief $P(S_{-1})$: first, we need to compute the probability of $S_0$ taking the value of $s_0$ as $P(S_0 = s_0) = \sum_{\forall s \in \Omega(S)} P(S_{t+1} = s_0 \mid S_t = s) P(S_{-1} = s)$; then, the probability of the entire trajectory can be computed multiplying the appropriate entries of the transition

model.

$$\Upsilon_1 = P(s_0, \ldots, s_T) = P(S_0 = s_0) \cdot \prod_{i=1}^{T} P(S_{t+1} = s_i \mid S_t = s_{i-1}) \tag{7.32}$$

The computation of this latter factor, $\Upsilon_2$, is the trickiest: it can be performed efficiently using a dynamic programming technique derived from the forward algorithm as explained in [41]. This algorithm iteratively computes the quantity $P(o_0, \ldots, o_T)$—from now on re-written as $F(T, s_T)$—using the transition and sensor models. The initialization step of the Forward algorithm is:

$$\forall x \in \Omega(S) \quad F(1, x) = P(S_0 = x) \cdot P(O_0 = o_0 \mid S_0 = x) \tag{7.33}$$

The distribution of $P(S_0)$ can be computed just like we did for Equation 7.32. The iteration step of the Forward algorithm is:

$$\forall x \in \Omega(S) \quad F(t+1, x) = \sum_{y \in \Omega(S)} F(t, y) \cdot P(S_{t+1} = x \mid S_t = y) \cdot P(O_t = o_1 \mid S_t = x) \tag{7.34}$$

Finally, having iterated the algorithm until $T$, the inverse of $\Upsilon_2$ is computed as the sum over all possible values of $S_T$:

$$\frac{1}{\Upsilon_2} = P(o_0, \ldots, o_T) = \sum_{\forall x \in \Omega(S)} F(T, x) \tag{7.35}$$

With the values of the three factors $\Upsilon_0$, $\Upsilon_1$, and $\Upsilon_2$, we can finally compute the conditional probability of the assignment of a trajectory of states, given the assignment of a trajectory of observations as:

$$P(s_0, \ldots, s_T \mid o_0, \ldots, o_T) = \Upsilon_0 \cdot \Upsilon_1 \cdot \Upsilon_2 \tag{7.36}$$

The detailed analysis of the time- and space-complexity of the computation of all three factors is given in the main body of this article: the most relevant result is the fact that the overall time-complexity is linear w.r.t. the time horizon $T$.

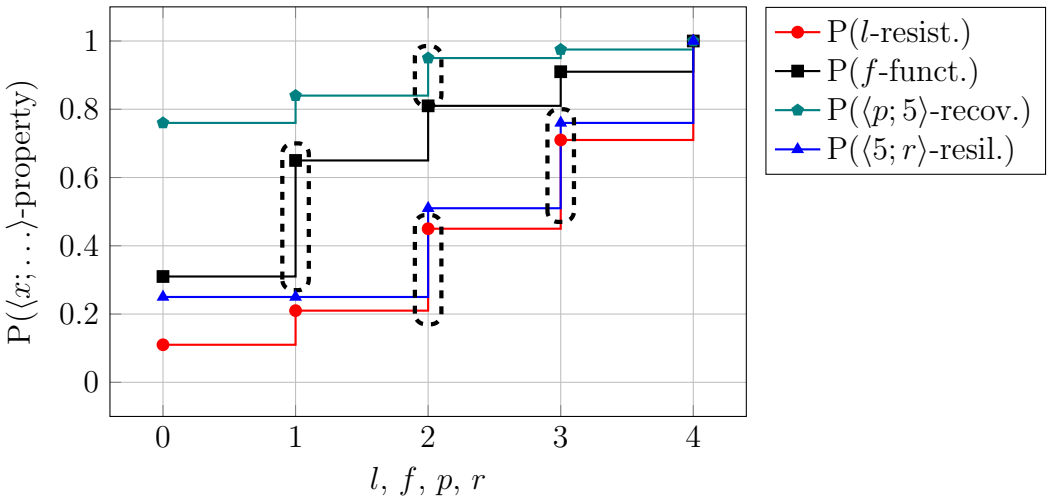



Figure 7.8 Probability distribution of the parametric resilient properties in a template scenario where $\forall s,\ c(s) \in [0, \ldots, 4]$. The discontinuities reveal the potentially critical thresholds for different properties.

# CHAPTER 8 ARTICLE 5 – FROM SWARMS TO STARS – TASK COVERAGE IN ROBOT SWARMS WITH CONNECTIVITY CONSTRAINTS

**Preface:** Finally, this chapter summarizes the work conducted over the last year of my doctoral program with the objective of rounding off my research with a practical application in the context of autonomous robotics. In particular, we were interested in investigating ways to autonomously and reliably maintain connectivity within a swarm of robots that can only rely on local communication and do not have access to a global positioning system. These are assumptions that we often encounter in the context of harsh environments and space expeditions—making multi-robot planetary exploration the ideal use case of the developed methodology. A significant discovery that we made while reviewing the literature—and trying to replicate its results—was that a majority of the previously proposed approaches were either lacking quick convergence or reliability. Because of this reason, we introduce a distributed controller that uses local interactions to drive tens of robots towards their targets—while in connected geometries—within minutes. As the main author of the article, my contributions included: reviewing (and re-implementing) several works in the literature on multi-robot connectivity, designing a new distributed control algorithm, implementing it in Python and Buzz, validating it in multiple ARGoS simulations, analyzing the data, preparing the figures, writing most of the document, and reviewing it in its entirety.

**Authors:** Jacopo Panerati, Luca Giovanni Gianoli, Carlo Pinciroli, Abdo Shabah, Gabriela Niculescu, and Giovanni Beltrame

**Submitted To:** Autonomous Robots – Special Issue on Distributed Robots: From Fundamentals to Applications

**Abstract:** Swarm robotics carries the potential of solving complex tasks using simple devices. To do so, however, one must be able to define a distributed control algorithm capable of producing globally coordinated behaviours. In this work, we propose and validate a methodology to address the problem of the spatial coverage of multiple tasks with a swarm of robots that must not lose global connectivity. Our methodology comprises two layers: at the bottom, a distributed Robot Navigation Controller (RNC) is responsible for simultaneously guaranteeing connectivity and pursuit of multiple tasks; on top, a global Task Schedule Generator approximates the optimal strategy for the RNC with minimal computational load. Our

contributions include: (i) a qualitative analysis of the literature on distributed multi-robot connectivity maintenance, (ii) the implementation of the proposed methodology, (iii) simulations performed in a multi-physics environment, and (iv) formal and experimental assessment of the guarantees on connectivity, coverage optimality, and fault-tolerance.

## 8.1 Introduction

Declining costs and thriving popularity are making drones and small robots a recurring sight in our daily lives. This trend, in turn, increases the appeal of multi-robots systems and swarm robotics in particular. Swarm robotics carries the potential of solving complex tasks using simple devices—and make the whole greater than the sum of its parts. One of the major challenges on its way, however, is the definition and implementation of distributed control algorithms capable of producing intelligent, coordinated behaviours, while relying on partial, local, and possibly noisy information.

In general, swarm robotics uses controllers that make decisions using local information such as the positions of a robot's direct neighbours or of its visible tasks. Therefore, even more than traditional multi-robot systems, robot swarms are capable of executing complex tasks in a distributed fashion, with less reliance on hierarchies or centralization. This, in turn, shifts the focus on the distributed control algorithms, running as identical—but separate—instances on each of the swarm members.

Many practical application scenarios require a swarm to remain fully connected throughout its entire operational life, e.g., when carrying a payload too heavy for a single or just a few robots. The direct consequence, and desirable property, of global connectivity is that a communication path always exists between any two swarm members. However, guaranteeing the swarm's connectivity through a distributed navigation algorithm is not at all a trivial task. For example, when the swarm is influenced by multiple external forces, such as the attractive potentials of two far-away task locations, boundary robots might move in opposite directions and alter the swarm formation irremediably. The challenge lays in finding the right way to combine possibly conflicting goals—using only distributed decision making and local information. A distributed controller should be able to drive the swarm using both well-known flocking/formation behaviours and task execution policies. A promising strategy is represented by control algorithms that exploit both flocking and expansion virtual potential functions associated to the position of the neighbouring robots, such as the Lennard-Jones potential [21].

In this work, we propose a hybrid methodology to address the problem of the spatial cov-

erage of multiple tasks by a swarm of robots that never loses global connectivity. Typical applications of this approach would be: (i) the autonomous exploration demanded by extreme and hard-to-reach environments [82]; or (ii) the fast deployment of a communication infrastructure or a GIS in an emergency area [7].

Our results—obtained from both mathematical modelling and multi-physics simulations—demonstrate that the modular and distributed nature of these multi-robot systems, on one hand, offers great potential for the development of autonomous and fault-tolerant mechanisms, but on the other, can aggravate the challenges posed by problems that are often practically intractable even in their centralized formulations (e.g., spatial coverage with the minimal Steiner tree [53]). The methodology in this work comprises two layers: at the bottom, a fully distributed Robot Navigation Controller (RNC) is responsible for simultaneously guaranteeing connectivity and pursuit of multiple tasks; on top, a global planner (the Task Schedule Generator) approximates an optimal strategy for the RNC with minimal computational load.

The rest of this article is organized as follows: Section 8.2 reviews works in the areas of task scheduling, task mapping, and swarm connectivity (comprehensive of a quantitative comparison); Section 8.3 describes the two-layer proposed methodology and its mathematical foundations; Section 8.4 and Section 8.5 present the settings, the results, and the discussion of our experimental evaluations. Finally, Section 8.6 concludes the article and suggests directions for the further development of the work.

## 8.2 Literature Review

The organization of this section mirrors the two-folded approach of Section 8.3. First, we review previous works tackling the problems of task scheduling, mapping and coverage that are addressed in the latter part of our methodology section. The second and larger part of this section, is instead dedicated to the problem of distributed assessment and control/maintenance of a swarm's connectivity, and it includes a critical review of previous works (corroborated by reimplementations and experimental comparisons).

### 8.2.1 On Task Scheduling, Mapping, and Coverage

Literature on the Multi-robot Task Allocation (MRTA) problem is exhaustively analyzed in [108]. Among MRTA literature, great relevance is given to temporal and ordering constraints for task execution [135], [91], [87], [107], [74], and [57]. Different objectives function have been proposed, including robot path distance minimization, total duration minimization,

and utility maximization [108]. Mixed-Integer Linear Programming (MILP) formulations for different variants of MRTA have been proposed in [57], [91], [87], [134]. In particular, [57], [87] have based their approaches on the centralized heuristic resolution of the proposed MILP formulation.

In [57], both geo-spatial and connectivity elements are ignored, while in [87] tasks have physical locations that the robots must reach, but the authors still do not address inter-robot communication. In [74], the robots communicate to determine the information available to each and run instances of a distributed task scheduling algorithm. The Consensus-Based Bundle Algorithm (CBBA) proposed in [135] aims at computing a task scheduling plan and correcting the schedule by driving some robots to behave as relays. In [13], task allocation is interpreted as a specific swarm behaviour. In a foraging scenario, probabilistic- and threshold-based methods are used to determine, in a distributed fashion, whether a robot should rest or should go out of the nest to collect objects. Connectivity constraints are not considered. To the best of our knowledge, this work is the first to propose a task scheduling and assignment problem for swarms of robots that jointly includes temporal, geospatial and connectivity constraints and couples it with distributed lower lever controller.

### 8.2.2 On Swarm Connectivity

Given a swarm of robots that can only communicate with their neighbours (i.e., within a limited range in a two-dimensional or three-dimensional space), we define (global) *connectivity* as the boolean property that tells us whether a communication path can be established within any two robots. This property is often desirable for its implications (e.g., the ability to implement distributed agreement through average consensus [172] or to continuously rely information from a specific robot to the entire swarm) and its distributed assessment and enforcement have been the objective of several previous research works [17], [89]. Besides the intrinsic challenges presented by distributed implementations, connectivity assessment and control can be made even more difficult in time-varying topologies, that is with robots that move or are affected by failures.

#### Eigenvector Centrality and Spectral Methods

The first category of research works that we examine is based on the formalisms of Spectral Graph Theory (SGT)[33]. Approaches in this class describe generic multi-robot systems as graphs $\mathcal{G}=(V,E)$ in which $K=|V|$ robots are represented by nodes and the existing communication links with non-directional arcs $e \in E$ (see Figure 8.1). This graphs, in turn, can be represented with the aid of matrices, in particular the adjacency matrix $A$ and the Laplacian

matrix $L$, tied by the following relationship:

$$L_{K\times K} := D - A = \begin{bmatrix} d_1 & 0 & \dots \\ 0 & d_2 & \dots \\ \dots & & \ddots \end{bmatrix} - \begin{bmatrix} 0 & a_{12} & \dots \\ a_{21} & 0 & \dots \\ \dots & & \ddots \end{bmatrix} \tag{8.1}$$

where $a_{ij} \in [0, 1]$ expresses whether two robots are neighbours, in other words if a link $e_{ij}$ exist between them and $D$ is the degree matrix, $d_i$ being the number of neighbours of $i$. The spectral analysis of the Laplacian matrix (the calculation of its eigenvalues and eigenvectors) reveals powerful insights about the underlying robot network. In particular, the second eigenvalue $\lambda_2$ of $L$ represents an upper bound of the sparsest (i.e., with the fewest links and separating the largest smaller partition) cut of the robot network, while the signs of the values in the second eigenvector of $L$ tell us on which side of this cut each robot would lie.

SGT can be used as a tool to discover how many link failures/disconnections a robot swarm can sustain before losing connectivity. However, it is important to keep in mind that SGT is not necessarily the be-all and end-all of swarm robotics connectivity: certain desirable but connected geometries (e.g., a line of robots) present many sparse cut opportunities. In the following, we review works that implemented spectral analysis in a distributed fashion (with a few caveats).

**Distributed Power Iteration Methods** SGT methods based on power iteration (PI) start from the observation that the repetition of the following update:

$$\mathbf{x}^{i+1} = M\mathbf{x}^i \tag{8.2}$$

allows to compute the largest eigenvalue (and associated eigenvector) of a generic matrix $M$ from a random initialization of $\mathbf{x}^0$. Furthermore, the PI of $L$ (and matrices directly derived from it) can be computed in a distributed fashion in the form of:

$$x_k^{i+1} = L_{kk} \cdot x_k - \sum_{j|L_{kj}=-1} x_j^i \tag{8.3}$$

[17] and [37] suggest different ways to derive a matrix $M$ from $L$ for Equation 8.2 so that each node in a network computes the second eigenvalue $\lambda_2$ of $L$ and its value of the associated eigenvector. However, even these approaches are not perfectly distributed: using Equation 8.3 still requires non-local information to be share to perform periodical normalization steps and avoid numerical instability. In [17] a beacon node is necessary to aggregate and broadcast

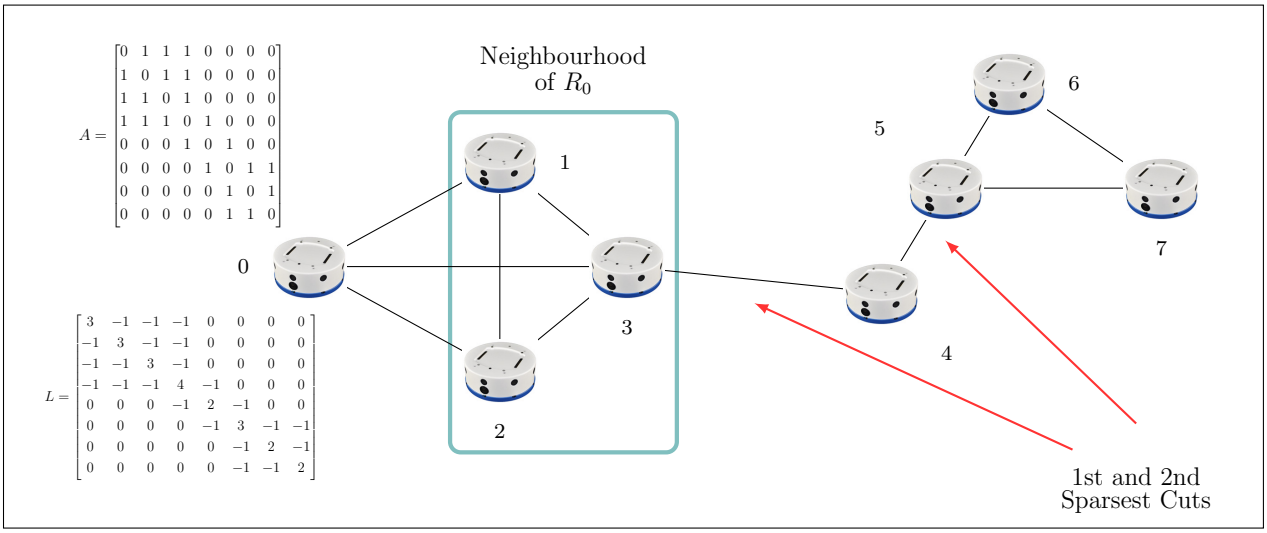



Figure 8.1 Robot swarms are often treated as graphs $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ (e.g. *via* Spectral Graph Theory) for networking purposes.

the norm of the second eigenvector that is necessary to normalize the result of the PI step; in [37] additional rounds of average consensus—potentially many times steps each—are used for the same reason. Other works, such as [176] and [59] extend the SGT discussion with the computation of multiple eigenvalues and topology changes, respectively, but are also limited in their performance by need to periodically agree on certain estimates across the whole swarm through, consensus.

**Wave Propagation-based Methods** Because of their frequent recourse to beacon nodes and or rounds of consensus, PI methods can hardly be considered fully distributed. Wave propagation-based methods resort to memory to circumvent the same issue. The method described in [145] can potentially find all eigenvalues associated to the Laplacian matrix $L$ of a robot network from the Fast Fourier Transform (FFT) of the signal propagated by each robot $i$ using:

$$s_i(t) = 2 \cdot s_i(t-1) - s_i(t-2) + k^2 \sum_{j|L_{ij}=-1} s_j(t-1) \tag{8.4}$$

where $k$ is a constant. The major drawbacks of this approach, however, are the relatively high requirements in terms of memory (each robot has to remember the history of $u$), possibly computation (each node should implement a FTT), and the sensitivity to accurate peak detection (in the FFT) that make it less suitable for the use with small robots, as it is often the case in swarms robotics, and noisy environments.

### Experimental Comparison of Spectral Methods

Based on the information found in [17], [145], and [37] we used the MATLAB-like scripting language Octave (the same used in some of the original implementations [17]), to compare the number of iterations that are necessary for these methods to converge to a precise estimate of $\lambda_2$. As error metric we chose the percent offset with respect to the actual value of $\lambda_2$, i.e., $e = |\dot{\lambda_2} - \lambda_2|/(10^{-2}\lambda_2)$. Our implementations are available under the MIT license on GitHub [1]. Our results are presented in Figure 8.2. As we would expect, the approach that most often requires consensus to normalize the results of its power iteration *PI2* ([37]) is the one that displays the slowest convergence. *PI1* [17] and *Wave* [145] are remarkably (and comparably) faster, with the first being more precise. To consider realistic applications, we then evaluated these approaches under the assumption of random packet drop, with probability $p$, on each of the inter-robot links. In this case, the results show that the slowest but more conservative methodology *P2* is the one most resilient to these errors (but only after $\sim 10^3$ iterations). Yet, all approaches are noticeably underperforming.

These results suggest that SGT—despite the appeal of its neat mathematical formulation—might not be be ideal approach to preserve swarm connectivity for two major reasons: (i) its slow convergence (hundreds of iterations even in small sized $K = 10$ swarms), and (ii) its sensitivity to noise. As a third reason we could add the fact that *per se* SGT only provides insights of the level of connectedness of a network, but this does not translate directly into a control strategy to maintain it.

### Biomimicry and Heuristics

Biomimicry behavioural and optimization algorithms (flocking, ant colony, PSO, etc.) are unsurprisingly popular in the context of swarm robotics [163], [69]. Most of these methodologies fall in the heuristic category as they typically approximate optimal solutions but do not provide any guarantees.

The approach presented in [89], in particular, superposes a collection of virtual forces (from which the control of each robot is computed) to drive a swarm of robots towards diverging leaders/tasks while also attempting to maintain the connectivity of the group. The primary components of this control are (i) a three-fold flocking algorithm, and (ii) attraction to the leader robots. As the authors discover from their experimental results, these components do no always suffice to maintain connectivity. They then add contributions named (iii) thickness, and (iv) density corrections. While the problem in [89] is very similar to that of

1. git@github.com:JacopoPan/ar-spectral-graph-theory-comparison.git

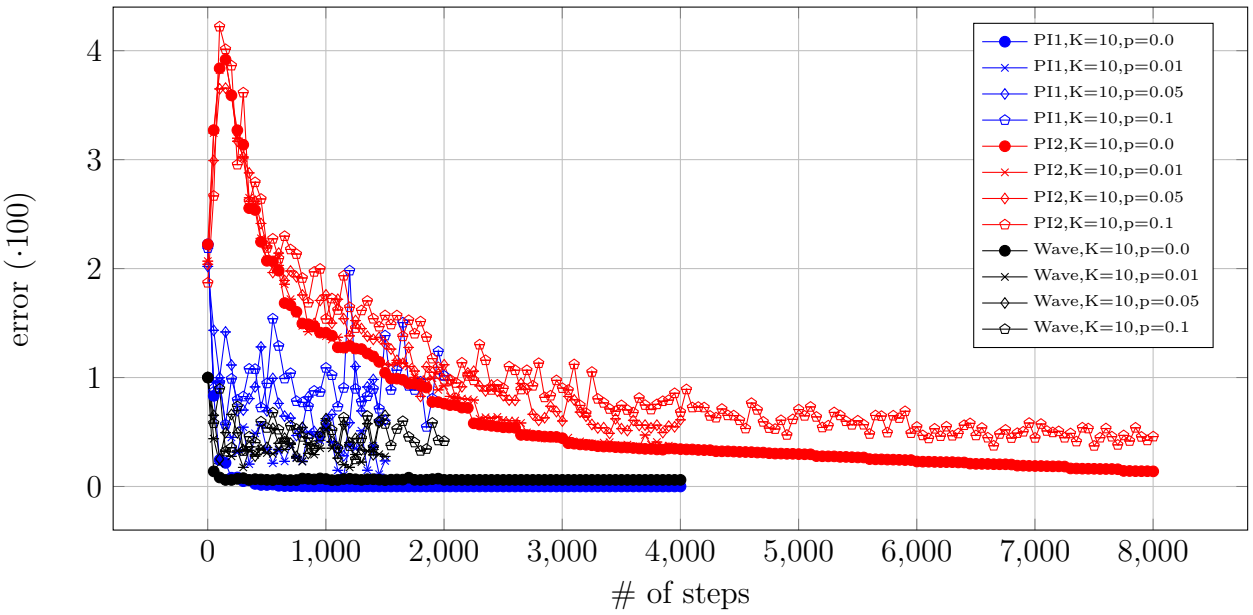



Figure 8.2 Comparison of the performance of Spectral Graph Theory methods for the computation of the second eigenvalue of the Laplacian matrix $\lambda_2$ under the assumptions of perfect communication packet drop with probability $p$.

spatial coverage of multiple tasks with a connected swarm that we want to approach here and it is based on a (partially) distributed control, it is not free of flaws: the number of control contributions requires impractically expensive parameter-tuning and, yet, there is no guarantee that the swarm will not eventually break down. In fact, even a controller finely tuned for a certain scenario might not perform equally well in another geometry.

**Flocking with Leader Forces** For the sake of completeness, we experimentally evaluated the combination of a three-pronged flocking algorithm, boundary tension, and leader attraction inspired by [89]. A Python Jupyter Notebook implementation of this approach is available under the MIT license on GitHub [2]. What we discovered is that, the higher the number of contributions to the control, the more parameter tuning is necessary. This might render a controller suitable for a certain geometry, but ill-fit for another. Furthermore, when leaders escape the swarm too quickly (or move too far), connectivity is eventually always lost, as shown in Figure 8.3.

2. git@github.com:JacopoPan/ar-flocking-and-leader-forces.git

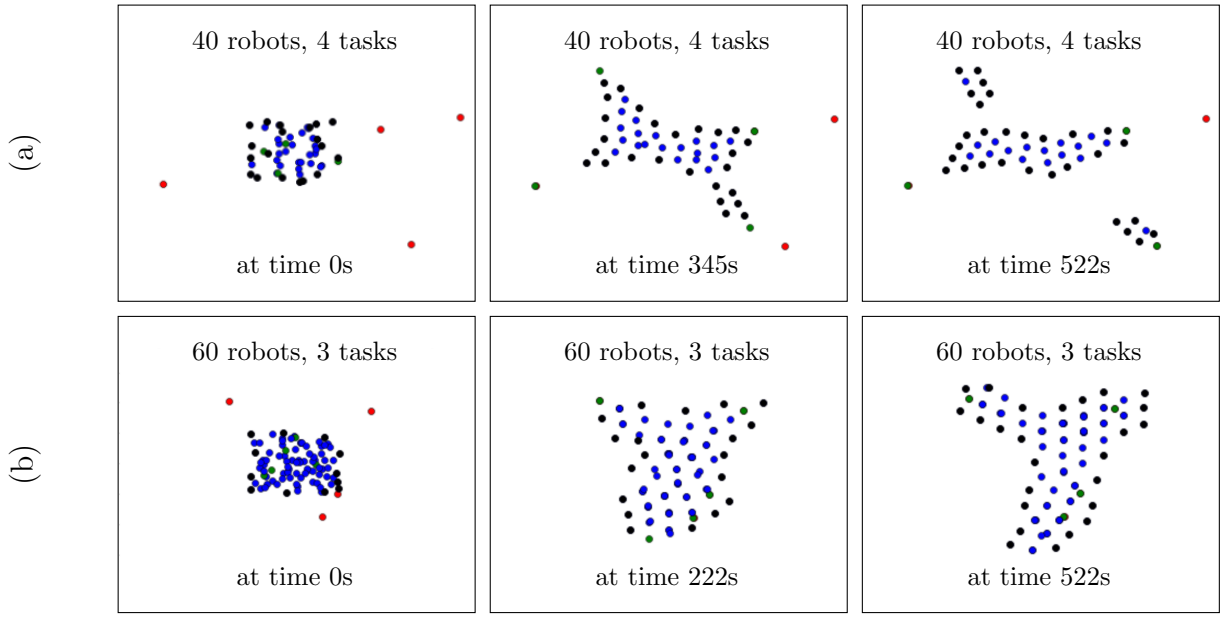



Figure 8.3 Superposition of flocking and leader forces in example scenarios with four (a) and three tasks (b), the simulation shows that a successful control strategy for one scenario does not necessarily generalize to others.

### Chaotic Oscillators, Receding Horizon Control, Resilient Formation, Etcetera

The distributed estimation of global topology changes is a problem that partially resembles that of distributed connectivity assessment. The approach in [18] exploits the fact that all the robots in the swarm are provided with a chaotic oscillators [125] whose synchronization properties in the context of graphs and networks have been studied by [126], [93]. The caveat here is that not all topology changes imply changes in the connectivity of a swarm. An Octave implementation of synchronizing discrete chaotic oscillators is available under the MIT license on GitHub [3].

The work in [157] describes a multi-layered controller in which connectivity of a multi robot system is enforced by middleware capable of refusing movements that would break an established communication graph (small changes are allowed over time). Other works investigate the incremental construction of networked multi-robot system that are resilient to the malicious behaviour of certain nodes [146]. This, however, it is partially beyond the scope of our research: in this work, when we address fault-tolerance, we refer to the ability to cope with packet-drop and robot failures rather than adversary behaviours.

3. git@github.com:JacopoPan/ar-discrete-chaotic-oscillators.git

## 8.3 Methodology

In this section, we present the details—from idea to implementation—of our methodology, as well as its formal and mathematical foundations. We start by providing a brief overview of our goal, the overall control architecture, and the two layers that compose it. We continue, in Subsection 8.3.2, with the description of three collective behaviours implemented by the means of a distributed navigation controller. Finally, in Subsection 8.3.3, we introduce a global planning layer what we use to prove how we could coordinate the distributed behaviours towards pseudo-optimal solutions.

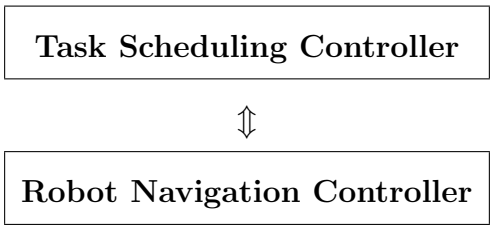
### 8.3.1 General Overview

Our goal is to drive a swarm of robots to cover multiple spatially distributed tasks without losing global connectivity, i.e., avoiding situations such as the one presented in Figure 8.3(a). For the sake of maintaining swarm connectivity, it is extremely useful to identify two classes of swarm members, i.e., *backbone robots* and *master robots*. The former set's main purpose is to keep the swarm connected; backbone robots move according to potential functions based on their neighbours positions or the indications of master robots. By contrast, master robots can perform special tasks such as (i) influencing backbone robots to move towards specific tasks and (ii) drive the whole swarm towards the locations of certain deployment points. [89] proposed a distributed algorithm that drives backbone robots to create trees (sub-optimal approximation of the ideal Steiner tree) whose leaves are represented by master robots. This method resembles our in its objectives but it is fundamentally different in how it attempts to reach them. In the approach of [89], a crucial issue is represented by the impossibility to prevent the master robots from excessively stretching the swarm—and thus breaking it—as they drive towards their assigned task locations.

These sorts of constraints are especially difficult to enforce in a distributed fashion, as each robot has no perfect knowledge on the position of the tasks. We only assume that master robots have approximate knowledge of the direction and distances of the tasks from each deployment point. A backbone robot is not able to determine, using solely its local information, whether it should stop to avoid the swarm disconnection or continue to move towards the task location. Global position information is typically hardly available to all robots for theoretical and practical reasons. Communication constraints arising from the use of tens or even hundreds of robots—for example, propagation delay over multi-hop communication paths, path explosion (for $N$ robots, $N^2 - N$ communications channels may be activated at the same time), and packet losses—may render impossible for a robot to communicate its

position to the others through multiple multi-hop ad-hoc wireless paths in a timely fashion.

In Figure 8.3, the fractures of the swarm were jointly caused by diverging motions and an insufficient number of backbone robots. To prevent this, we chose to address the problem from a new perspective: pushing robots from well-known deployment points rather than pulling towards unknown tasks. We introduce a swarm task scheduling layer as one of the entity responsible for swarm integrity. Then, we split the swarm control architecture into two layers:




At the bottom, the distributed Robot Navigation Controller allows each swarm member to keep a behaviour coherent with its role within the swarm, i.e., that of a master robot or a backbone robot. The RNC leads each master robot to an assigned position, while smartly adjusting the position of the backbone robots to build a reliable (w.r.t. communication failures) robot structure (network) connecting the master robot to the tasks locations. The RNC is practically implemented through a distributed navigation algorithm that runs independently—as identical instances—on each swarm member. The RNC offers to the upper layers a *best effort connectivity service*, i.e., it guarantees that it will do all that it can to keep the swarm connected.

The TSC is responsible for selecting, at any given time instant, (i) which subset of tasks should be executed by the swarm in a simultaneous fashion, (ii) which robot should be elected master robot, and (iii) how many tasks should be sought from each deployment point. To complete a task, backbone robots are demanded by the master robot to navigate towards the location of the task itself (without necessarily knowing the distance they will have to travel). Swarm connectivity is guaranteed by the TSC by the fact that it only considers task subsets that the RNC shall be able to successfully support (i.e., connecting the task locations in a reliable manner).

In this architecture (see Figure 8.4), the more efficient the RNC, the easier the task the TSC. As, the capability of the RNC to cover a subset of task locations depends on (i) the number of available backbone robots, (ii) the communication range of each robot, (iii) the position of the tasks, and (iv) the efficiency of the RNC. The capabilities of the RNC are hard constraints that have to be considered by the task scheduling optimization problem solved by the TSC.

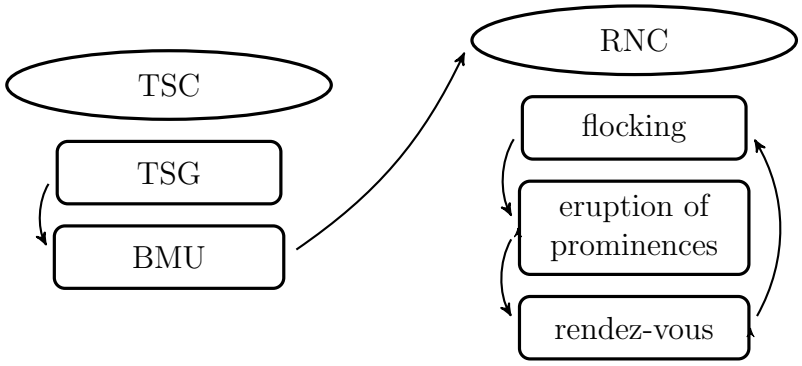



Figure 8.4 Flowchart of the overall control architecture.

### 8.3.2 Robot Navigation Controller

The Robot Navigation Controller is the distributed navigation module that, by controlling each robot independently, produces and coordinates different collective swarm behaviours. We consider a RNC supporting the following three swarm behaviours:

— **Flocking**: the swarm travels toward a new destination while maintaining a compact formation around a single master robot. Once there, the master robot is assigned to the task called Swarm Centroid (SC). During the swarm displacement, potential functions and consensus drive all the other swarm members, i.e., the backbone robots to pursue the moving master robot while conserving the original formation. For flocking to be successful in large swarms, homogeneous and compact disposition (as introduced by the following behaviour) is a must.

— **Rendez-vous**: the swarm regains its compact formation around a single master robot assigned to the SC task. This behaviour is used just before a flocking phase (and just after). The fixed location of a master robot represents the center of the swarm formation. This behaviour requires the swarm to be fully connected at the time of the activation.

— **Eruption of Prominences** [4]: this behaviour allows the backbone robots to reach distant targets without losing connectivity with the SC. The swarm lays in a fixed formation around the master robot assigned to the SC task. According to the indications received by the TSC layer, the master robot pushes the backbone robots to travel towards different tasks. Each backbone robot is assigned to a single task. The master

4. We derived our terminology from the loose resemblance that these structures have with the Solar surface phenomena. Hence the title of this article.

robot assigned to the SC keeps its original function. Backbone robots automatically adjust their position and speed with respect to neighbouring robots to maintain the connectivity of the swarm.

Through the proposed implementation of the third behaviour, RNC enables the ability to pursue multiple tasks (directions with respect to the SC) without relying on a global positioning system nor the need to run consensus steps (unlike, e.g., PI-based SGT methods). Furthermore, we show that the link redundancy of the directional eruptions is indirectly parametrized by the use of potential functions, making them tolerant to failure—any robot can replace another one seamlessly because it implements the same controller that only responds to visible neighbours. Finally, during the eruption of prominences, connectivity is implicitly but strongly enforced: because backbone robots are pushed from the center by potentials—rather than being pulled from the tasks—their control does not contain components that might drive them to disconnect.

### Robot Model

All the behaviours implemented by the RNC are based on a robot model that makes very loose assumptions. These are: (i) robots have identical communication and movement capabilities (with a maximum speed $v_{max}$ and constant mass $m_r$); (ii) they have a limited communication range $r$; (iii) no information on global positioning (GPS, GLONASS, Galileo, nor BeiDou); and (iv) they can exploit the situated communication model [160]—that is, the ability to assess distance and bearing of the other robots they speak to.

### Lennard-Jones Potential

The Lennard-Jones potential (see Figure 8.5) is a model of inter-atomic interaction that finds frequent use in the context of robotic interaction [21]. We briefly reintroduce it here as it is exploited as a low-level component by all of our RNC behaviours. The Lennard-Jones potential provides a smooth combination of attractive and repulsive forces that can be used to homogeneously diffuse robots from random initial positions.

Its advantages are: (i) the simple and distributed math; and (ii) the stable, smooth and predictable equilibrium. Its convergence can be very slow in certain implementations. The

potential (and derived force contribution) equations are as follows:

$$P_{LJ} = \epsilon \left( \left( \frac{\delta}{x} \right)^a - 2 \cdot \left( \frac{\delta}{x} \right)^b \right) \tag{8.5}$$

$$F_{LJ} = -\epsilon \left( \left( \frac{a \cdot \delta^a}{x^{a+1}} \right)^a - 2 \cdot \left( \frac{b \cdot \delta}{x^{b+1}} \right)^b \right) \tag{8.6}$$

The parameters $\epsilon$ and $\delta$ represent the depth of the minimum in the potential and its distance from the origin, respectively. The exponents $a$ and $b$ are typically set to 12 and 6 but, in our implementation, we used 4 and 2 for an easier control.

### Flocking

The RNC implements the flocking behaviour to allows the swarm to collectively move from one point of deployment (SC) to another. Distributed approaches to flocking are well established in the literature. Based on the approach proposed in [139], we derived:

$$v_i^{t+1} = \sum_{j \in \mathcal{N}} \left( c \cdot x_j^t + a \cdot v_j^t - s \cdot \Gamma(d_{ij}^t < t) \cdot d_{ij}^t \cdot x_{ij}^t \right) \tag{8.7}$$

where $v_i^{t+1}$ is the new velocity of robot $i$ at time $t+1$, $\mathcal{N}$ is the set of neighbours of $i$, $x_j^t$ and $v_j^t$ are the position and velocity of robot $j$ at time $t$, $d_{ij}^t$ is the distance between $i$ and $j$, $t$ a threshold, $\Gamma$ a function that evaluates to 1 if its input condition is met, and $c$, $a$, $s$ the cohesion, alignment, separation coefficients. Then we combine the contributions of Equations 8.6 and 8.7 to create a distributed controller that lets all robots move together while also spreading homogeneously at predictable inter-robot distances to improve connectivity.

### Rendez-vous

The RNC resorts to the rendez-vous behaviour in the two following situations:

1. When the swarm re-group after a flocking phase and before erupting into prominences.
2. When the swarm re-group from an eruption phase and before flocking towards another deployment point.

In practice, this behaviour is achieved through the broadcast—by the master robot—of two messages, one containing a new, smaller $\delta$ parameter to use in Equation 8.6 and a second message forcing all backbone robots to solely base their control on Equation 8.6. Backbone robots that receive these messages, further relay the information so that robots not directly

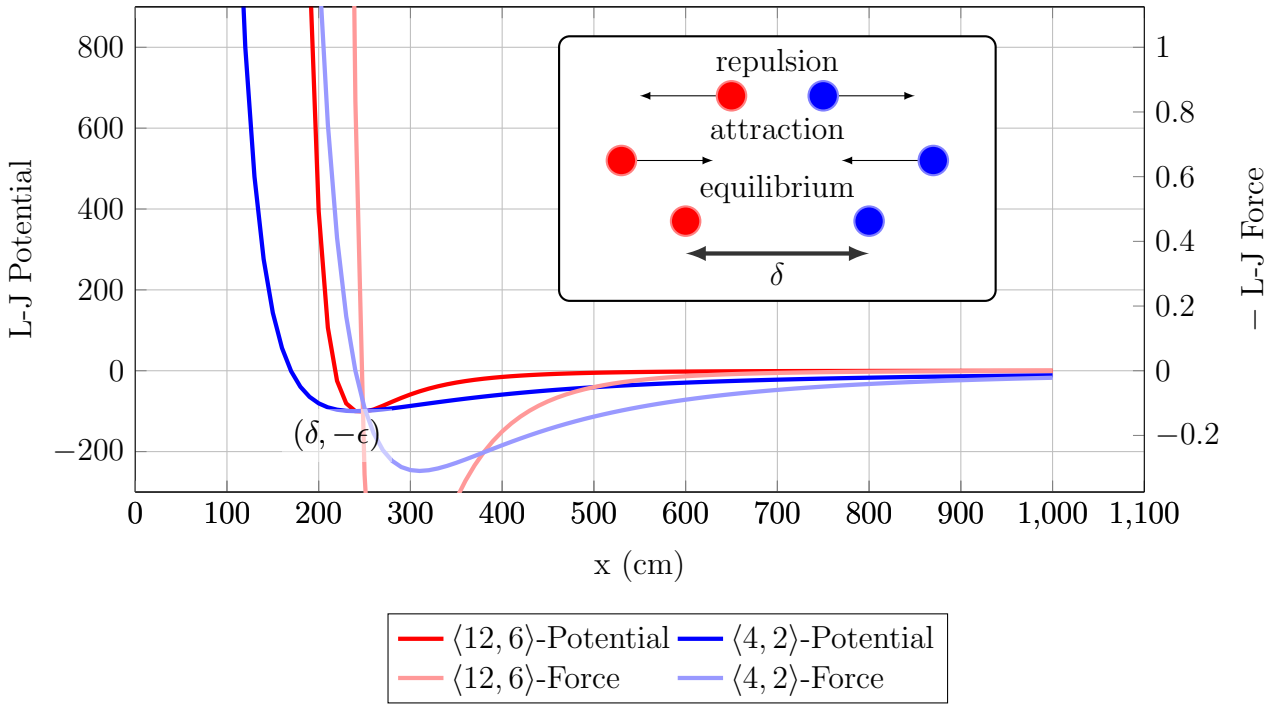



Figure 8.5 The Lennard-Jones potential (and the force derived from it) is used in our RNC to regulate attraction and repulsion between neighbouring robots. Typically, the exponents used in its computation are 12 and 6, in our implementation we use a smoother function with exponents 4 and 2.

in sight of the master robot are also affected. The smaller the value of $\delta$, the more compact the resulting swarm.

### Eruption of Prominences

In [89], the expansion algorithm exploits multiple attraction and repulsion forces to lead backbone robots to build a pseudo-Steiner tree connecting the diverging master robots traveling toward their corresponding task locations. Although the ideal Steiner tree is the most efficient way of connecting a desired set of points (i.e., the task assigned to the master robots) while using the overall shortest set of segments (and, therefore, the minimum number of auxiliary the backbone robots, assuming uniform spacing), we noticed that the proposed tree expansion algorithm is not necessarily optimal (that is, converging to the Steiner tree) and very sensitive to a significant number of input parameters. This makes the algorithm unreliable, and thus unfit for general and widespread use in *a priori* unknown scenarios, such as disaster relief.

In this paper, we propose a new expansion algorithm, the Star Eruption for Connected Swarms (SECS), that, as the name suggests, leads the backbone robots to form star-like formations that connect several tasks to a central robot, each with a dedicated arm. Given a subset of tasks to be simultaneously accomplished, each backbone robot is randomly assigned the arm of a specific task with a probability related to the distance between the location of each task and the swarm center (known to the master robot).

Once assigned to a specific arm, each backbone robot is driven by potential functions and angular correction suggestions broadcast by all robots to their neighbours, to find its position within the arm.

Although a star-like formation is naturally less efficient than the optimal Steiner tree in terms of total lengths of the segments number of auxiliary points (the backbone robots) required to connect all the points (the tasks), SECS provides two major advantages. SECS provides: (i) higher reliability with respect to faults because any robot takes part in providing connectivity with at most one task (in a Steiner tree, instead, the loss of a single robot could imply the loss of connectivity to multiple tasks); and (ii) SECS is only influenced by a few input parameters that are not sensitive to the specific features of a scenario. The pseudo-code of SECS is presented in Algorithm 3.

**From a Distributed to a Shared Coordinate System** The first challenge in creating a coherent prominence (whose absolute direction is only known to the master robot) is the fact that each robot in the swarm possesses its own—and constantly moving—coordinate

systems. A robot can infer the angular displacement of its neighbours using the situated communication model but it does not know where they are pointing. In order to create collective agreement on the direction of the prominence to built, all robots $j$ that know the direction of the prominence in their own coordinate space (initially only the master robot) must propagate two messages: $\angle T_j$ (broadcast) with the local prominence direction and $\angle M_{ji}$ (sent from $j$ to $i$) with the local direction of $i$, as seen by $j$. Then, each robot $i$ can recompute the direction of the prominence in its own coordinate system as:

$$\angle\, T_i \leftarrow \angle\, M_{ji} - \angle\, O_{ij} + \angle\, T_j \tag{8.8}$$

where $\angle\, O_{ij}$ is the observation of the direction of $j$ in the coordinate system of $i$.

**Alignment Maneuver** As soon as a backbone robot learns, from the master or another robot, about the prominence direction $\angle T_i$, it should move towards it. However, this step requires a careful trade-off in between two conflicting goals: (i) letting all robots pursue the direction of the prominence on their shortest path and (ii) not overcrowding the neighbourhood of the master robot. If (ii) happens, the resulting collisions and communication interferences could degradate the performance of SECS or prevent the formation of the prominence altogether. The way we choose to implement this alignment maneuver is by letting each robot *spiraling* (towards $\angle T_i$) around the robot from which it lastly received information about $\angle T_j$ using a randomized parameter $\theta$ (to avoid robot collisions).

**Combination with the LJ Potential** As described in Algorithm 3, when a robot finally aligns with the prominence, its control is taken over by the Lennard-Jones potentials of its neighbours. The repulsive forces allow the prominence to erupt towards the desired direction, while the attractive ones prevent the break-down of the swarm. If a robot loses its alignment (e.g., because it was pushed by a neighbouring robot), its control goes back into the spiraling step.

**Real World Physics Shortcomings and Required Adjustments** The first step in the validation of SECS was its implementation in an idealized scenario devoid of troubles such as collisions, inertia, or packet loss. A Jupyer Notebook version of the Python code is available under the MIT license on GitHub [5]. Its performance is demonstrated by the experiments reported in Figure 8.6: two swarms of different sizes erupting into three prominences with bearings $-45°$, $45°$, and $90°$.

5. git@github.com:JacopoPan/ar-prominences-in-the-ideal-world.git

```
1  init_robot ;
2  while eruption = true do
3  |  if master = true then
4  |  |  read(tsc-plan);
5  |  |  broadcast(dir);
6  |  else
7  |  |  broadcast(local-dir);
8  |  |  if return-node() ≠ nil then
9  |  |  |  aligned = offset(return-node());
10 |  |  |  if aligned then
11 |  |  |  |  lj-potential(delta);
12 |  |  |  else
13 |  |  |  |  spiral(return-node());
14 |  |  |  end
15 |  |  else
16 |  |  |  lj-potential(regroup-delta);
17 |  |  end
18 |  end
19 |  send(status,messages);
20 |  read(messages);
21 end
```

**Algorithm 3:** Pseudo-code of SECS

The real world, however, is a much more complex environment and, for the controller we tested in the multi-physics simulator ARGoS, we extended the RNC with the following:

1. Memory, i.e., a knowledge base to store the most recent information about the observed neighbours and possibly cope with sudden disconnects.
2. A gradient—propagated from the master robot—to dynamically estimate the distance covered by a robot in the prominence and assess whether a neighbour is closer to the center or the extremity of it.

This second feature, in particular, can be exploited at line 8 of Algorithm 3 and sensibly improved the performance of ARGoS/Buzz simulations.

**Connectivity and Reliability** If not under exceptional circumstances (e.g., communication affected by packet drop with $p > 0.9$ or simultaneous robot failure), the proposed RNC methodology preserves the global connectivity of the swarm. All swarm behaviours are, in fact, implemented through contributions that never let robots move away from one another further than the parameter $\delta$ allows. Furthermore, the link redundancy $w$ of the swarm (and its reliability to node failures) can be tuned through the communication range $r$ and $\delta$

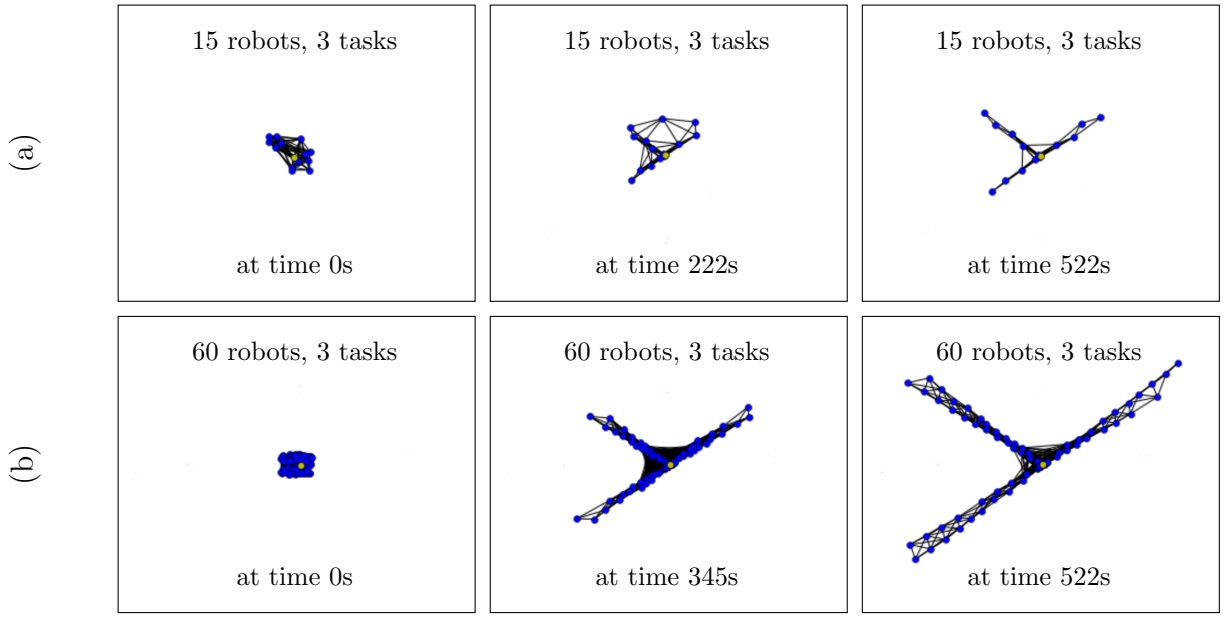
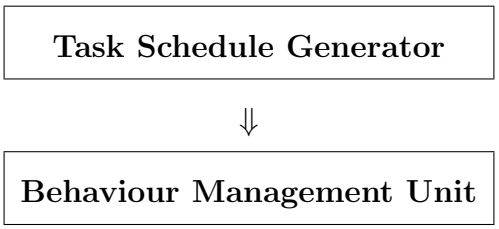



Figure 8.6 Simulation of the proposed prominence eruption algorithm towards three different tasks with 15 (a) and 60 (b) robots in an idealized model without collisions nor packet drop on the neighbour-to-neighbour communication channels.

parameter exploiting the inequality $w \geq r/\delta$.

### 8.3.3 Task Scheduling Controller

The Task Scheduling Controller is responsible for two main operations: (i) computing the subsets of simultaneous tasks that have to be executed in each specific time slot, (ii) managing the transition between successive time slots (and thus different subsets of tasks). The TSC can be logically represented by the two-layer architecture:




At the lower level lays the Behaviour Management Unit (BMU). The BMU receives from the Task Schedule Generator (TSG) a sequence of task subsets that have to be performed in multiple time slots (one time slot per task subset). The duration of each time slot is low-bounded by the execution time required by the longest task of the corresponding task subset.

Each task subset is uniquely defined by its tasks and the deployment point, the point that must be reached in closed formation by the whole swarm (that is, the master robot assigned to the SC task and the rest of the swarm flocking around it) before triggering the *eruption of prominences* behaviour that will lead the *backbone robots* towards the tasks.

The BMU elaborates the received task sequence to generate the chain of swarm behaviours for the underlying RNC. Let $(T_1, T_2)$ be a sequence of two task subsets, let $\Delta_1$, $\Delta_2$ represent the duration of the longest task for each subset and let $\Theta_1$, $\Theta_2$ be the corresponding deployment points. Assuming that the swarm starts the operations with a compact formation around the stationary master robot assigned to the SC task, the swarm behaviour sequence generated by the BMU would be:

1. **Flocking**, to displace the swarm in compact formation from the starting point to deployment point $\Theta_1$.
2. **Eruption of Prominences**, to drive the appropriate backbone robots toward the corresponding task locations for $T_1$.
3. **Rendez-vous**, after a time $\Delta_1$, to force the whole swarm to regain the original compact formation around the master robot assigned to the SC task placed in $\Theta_1$.
4. **Flocking**, to dislocate the swarm in compact formation from the previous deployment point $\Theta_1$ to the next deployment point $\Theta_2$.
5. **Eruption of Prominences**, to drive the appropriate backbone robots toward the corresponding task location for $T_2$.
6. **Rendez-vous**, after a time $\Delta_2$, to force the whole swarm to regain the original compact formation around the master robot assigned to the SC task placed in $\Theta_2$.
7. And so forth, for all the time slots, until the completion of all tasks.

At the upper layer, the TSG takes as input the whole set of tasks requiring completion at a given time and produce a sequence of task subsets for the BMU. The sequence generation must be optimized to reduce the overall execution time and the total distance traveled by the robots, while respecting the hard constraints on the swarm connectivity and reliability. The swarm connectivity constraints must reflect the way in which the swarm is organized by the expansion algorithm of the robot navigation controller. For instance, when the eruption of prominences is implemented through the proposed algorithm, swarm connectivity is guaranteed if a star-like formation connecting each task to the swarm center through a dedicated arm can be built reliably; the minimum number of robots that have to be assigned to each arm unit of length should be adjusted to the desired level reliability to failures. This aspects are explored thoroughly in Section 8.3.4.

In the following, we show how to formulate the mathematical model representing the optimal task scheduling problem that is solved by the TSG every time it generates a new task subset sequence. In particular, we discuss a mixed-integer linear programming (MILP) formulation that, with commercial state-of-the-art solvers like IBM CPLEX or GUROBI, can be near-optimally solved in the order of minutes even for large instances with up to 20 robots and 100 tasks.

### 8.3.4 Mathematical Modelling of the Optimal Task Scheduling Problem

The goal of the TSG is to find the optimal sequence of task subsets to pursue while also guaranteeing that the underlying RNC will succeed in keeping the whole swarm connected and resilient to robot failures. Let us consider a robot swarm operating in a convex region $R$. The convexity of the region is crucial to efficiently model the connectivity structure built by the algorithm of the RNC, e.g., the star constructed by SECS; a concave region would require a dramatic complexity increase to manage region discontinuities. Let $r^X_{max}$ ($r^Y_{max}$) and $r^X_{min}$ ($r^Y_{min}$) be the real parameters representing the maximum $X$ ($Y$) coordinate and the minimum $X$ ($Y$) coordinate that delimit region $R$, respectively. The set of robots in the swarm is denoted by $N$. The swarm is initially placed in compact formation around the deployment point of coordinates $\left(\Theta^X_0, \Theta^Y_0\right)$. Being $C$ the set of robot capabilities, e.g., `RGB-camera`, `infrared sensor`, `aerial`, the binary parameter $\phi^c_n$ is equal to 1 if robot $n \in N$ has capability $c \in C$. We also assume that all the robots are equipped with the same communication technology, that guarantees robot-to-robot communication within a maximum communication range of value $\Gamma$. Capability parameters $\phi^c_n$ are elaborated to define a set $K$ of configuration classes, where each class contains all the robots with the same set of capabilities, e.g., no special capability or `RGB-camera` plus `infrared sensor`.

At the moment of the optimization, the robot swarm is asked to accomplish the set of tasks $T$. A task $t \in T$ is characterized by the location coordinates, the list of required robot capabilities, the number of demanded robots, the expected task duration. Real parameters $\sigma^X_t$ and $\sigma^Y_t$ represent the $X$-$Y$ coordinates for the location of task $t \in T$. The binary parameter $\upsilon^c_t$ is equal to 1 if task $t \in T$ requires a robot with the capability $c \in C$. This parameter is exploited to define the new binary parameter $\tau_{tk}$, equal to 1 if a robot of configuration class $k$ is able to perform task $t$.

Finally, let $\beta_t$ be the number of robots required by task $t \in T$ and let $\delta_t$ be the expected completion time for task $t \in T$. A task priority scheme may also be enforced through binary parameters $\mu^{t1}_{t2}$, which are equal to 1 if task $t1 \in T$ has to be executed before task $t2 \in T \setminus \{t1\}$. We denote with $S$ the ordered set of consecutive time-slots (the ordering of the

slots implies that time-slot $s$ begins when time slot $s-1$ ends and so forth). Note that the integer parameter $a_s$ is equal to the position of time slot $s$ in the ordered set $S$. To guarantee reliability to robot failures or temporary communication impairments, it can be desirable to guarantee the existence of $\Omega$ robot-disjoint communication paths between each task and the central robot.

**Total Completion Time Minimization**

We exploit the aforementioned notation to formalize a MILP formulation for the Total Completion Time Minimization (TCTM) that guarantees SECS-based swarm connectivity throughout the entire duration of the scenario. Let $z^s$ be the non-negative real variables representing the duration of time-slot $s \in S$; note that $z^s$ is kept to 0 if a time-slot contains no task. Furthermore, let $x_t^s$ be the binary variables equal to 1 if task $t \in T$ is scheduled for execution in time-slot $s \in S$. The time minimization objective function can be expressed as:

$$\Lambda \;=\; \min \left\{ \sum_{s \in S} z^s \right\}. \tag{8.9}$$

The total duration $z^s$ of time-slot $s \in S$ is bounded from below by the expected completion time of the longest task scheduled for execution during that time-slot. This relation is expressed by:

$$z^s \geq \delta_t x_t^s, \qquad \forall s \in S,\, t \in T. \tag{8.10}$$

Each task must be scheduled in exactly one time-slot:

$$\sum_{s \in S} x_t^s = 1, \qquad \forall t \in T. \tag{8.11}$$

We introduce the non-negative real variables $y_t^{Xs}$ and $y_t^{Ys}$ to account for the $X$-axis and $Y$-axis distances between a task $t \in T$ and the swarm deployment point chosen for time-slot $s \in S$. Similarly, the $X$-$Y$ coordinates of the swarm deployment point for time-slot $s \in S$ are represented through real variables $g^{Xs}$ and $g^{Ys}$. The following groups of constraints are

included in the model to correctly compute the values of the distance variables $y$:

$$y_t^{Xs} \geq \sigma_t^X - g^{Xs}, \qquad \forall s \in S,\, t \in T, \tag{8.12}$$
$$y_t^{Xs} \geq -\sigma_t^X + g^{Xs}, \qquad \forall s \in S,\, t \in T, \tag{8.13}$$
$$y_t^{Ys} \geq \sigma_t^Y - g^{Ys}, \qquad \forall s \in S,\, t \in T, \tag{8.14}$$
$$y_t^{Ys} \geq -\sigma_t^Y + g^{Ys}, \qquad \forall s \in S,\, t \in T. \tag{8.15}$$

A task can be executed during a specific time-slot only if enough backbone robots are available to build an arm that connects the task and the master robot assigned to the SC task. The distance between the backbone robots in an arm must be smaller than the communication range $\Gamma$, and $\Omega$ robot-disjoint paths should exist between the edge and the center. To determine the maximum distance that can be covered by an arm, let us introduce the non-negative integer variables $w_t^s$ representing the number of robots that will form the arm of task $t \in T$ during time-slot $s \in S$. Ideally, the distance constraint should be expressed by a classic circle equation such as:

$$\left(y_t^{Xs}\right)^2 + \left(y_t^{Ys}\right)^2 \leq \left(\Gamma w_t^s\right)^2. \tag{8.16}$$

However, such non-linearity should be avoided to not dramatically increase the problem complexity. To this purpose, instead of bounding distance variables $y$ with a circle of radius $\Gamma w_t^s$, we approximate the positive quadrant of the circle through three linear pieces, i.e., the vertical line passing at $(\Gamma w_t^s, 0)$, the horizontal line passing at $(0, \Gamma w_t^s)$ and the diagonal line passing at $\left(\sqrt{\frac{\Gamma w_t^s}{2}}, \sqrt{\frac{\Gamma w_t^s}{2}}\right)$ with first derivative equal to $-1$. Note that more contributions could be used to improve the approximation. The three contributions are modelled through the three groups of constraints below. Note that a few additional modifications were added to these constraints: (i) a reliability parameter $\Omega$ divides the communication range to enforce the use of more robots and guarantee the presence of $\Omega$ robot-disjoint paths, (ii) a reliability parameter $\Omega$ is also added to $w_t^s$ to account for the special case when no backbone robots are required in the connectivity arm because the task location lays within the communication range of the swarm center, (iii) a big $M$ term that makes the constraints useless when the

task is not executed in the considered time slot. The constraints are expressed as follows:

$$y_t^{Xs} + y_t^{Ys} \leq \frac{\sqrt{2}\Gamma}{\Omega}\left(w_t^s + \Omega\right) + M\left(1 - x_t^s\right), \qquad \forall s \in S,\, t \in T, \tag{8.17}$$

$$y_t^{Xs} \leq \frac{\Gamma}{\Omega}\left(w_t^s + \Omega\right) + M\left(1 - x_t^s\right), \qquad \forall s \in S,\, t \in T, \tag{8.18}$$

$$y_t^{Ys} \leq \frac{\Gamma}{\Omega}\left(w_t^s + \Omega\right) + M\left(1 - x_t^s\right), \qquad \forall s \in S,\, t \in T. \tag{8.19}$$

Let $f_{kt}^s$ be the non-negative integer variables representing the number of robots of configuration class $k \in K$ elected as master robots in charge of task $t \in T$ during time slot $s \in S$. The following constraints prevent a task from being assigned to a master robot belonging to a configuration class without the required set of capabilities:

$$f_{kt}^s \leq \alpha_k \tau_{kt} x_t^s, \qquad \forall k \in K,\, s \in S,\, t \in T. \tag{8.20}$$

Furthermore, the number of robots required by a task must be satisfied during the time slot scheduled for the task completion. Note that a robot can be assigned to no more than one task per time slot:

$$\sum_{k \in K} f_{kt}^s = \beta_t x_t^s, \qquad \forall s \in S,\, t \in T, \tag{8.21}$$

$$\sum_{t \in T} f_{kt}^s \leq \alpha_k, \qquad \forall k \in K,\, s \in S. \tag{8.22}$$

The total number of robots required to execute a subset of tasks scheduled during the same time slot while keeping the connectivity, must not exceed the total number of swarm member $|N|$:

$$\sum_{t \in T} \left(w_t^s + \beta_t x_t^s\right) \leq |N| - 1, \qquad \forall s \in S. \tag{8.23}$$

Note that the explicit use of the configuration classes instead of considering the single robot-task assignment allows to significantly reduce the problem complexity (the number of configuration classes should be smaller than the number of robots). The following valid inequalities, stating that backbone robots are assigned only to arms of active tasks, can be included to reduce the solution space and speed up the solution computation:

$$w_t^s \leq \left(|N| - 1\right) x_t^s, \qquad \forall s \in S,\, t \in T. \tag{8.24}$$

Another group of constraints can be added to force the solution to use the first available time slot, therefore restricting the solution space:

$$\sum_{t\in T} x_t^s \leq |T| \sum_{t\in T} x_t^{s-1}, \qquad \forall s \in S. \tag{8.25}$$

Finally, a last group of constraints is included to respect the priorities defined by parameters $\mu$:

$$\sum_{s1\in S} a^{s1}\mu_{t2}^{t1} x_{t1}^{s1} \leq \sum_{s2\in S} a^{s2} x_{t2}^{s2} - 1, \qquad \forall t1, t2 \in T. \tag{8.26}$$

For sake of completeness, we also report the variables domains below:

$$z^s,\, y_t^{Xs},\, y_t^{Ys} \geq 0, \qquad \forall s \in S,\, t \in T, \tag{8.27}$$

$$g^{Xs},\, g^{Ys} \in \mathcal{R}, \qquad \forall s \in S, \tag{8.28}$$

$$w_t^s \in \mathcal{N}, \qquad \forall s \in S,\, t \in T, \tag{8.29}$$

$$\eta_{nt}^s,\, x_t^s \in \{0, 1\}, \qquad \forall n \in N,\, s \in S,\, t \in T. \tag{8.30}$$

**Total Dislocation Space Minimization**

Once the minimum completion time is minimized by the TCTM formulation ( 8.9- 8.30), we propose to address a second optimization problem to minimize the overall robot dislocation space while guaranteeing the optimal completion time previously computed. This kind of sequential optimization strategy is usually referred to as lexicographic: the $i$-th objective cannot be optimized at the expense of the $(i-1)$-th objective. Alternatively, while optimizing the $i$-th objective, the values of the $(i-j)$-th objectives $\forall j \in \mathcal{N}_{\geq 1}$ are considered as problem constraints. Let $\bar{S}$ be the time slot ordered set that does not include the first time slot and let $S_0$ be the time slot set containing only the first slot. Furthermore, let $\bar{g}^{Xs}$, $\bar{g}^{Ys}$ be the non-negative real variables representing the $X$ and $Y$ distances, respectively, between the deployment point of time slot $s \in S$ and that of time slot $s-1$; note that for $s \in S_0$, the distance is computed between the deployment point and the original starting point of coordinates $\left(\Theta_0^X, \Theta_0^Y\right)$ The objective function of the Total Dislocation Space Minimization (TDSM) problem is expressed by:

$$\min \left\{ |N| \sum_{s\in\bar{S}} \left(\bar{g}^{Xs} + \bar{g}^{Ys}\right) + \sum_{s\in S, t\in T} \left(\frac{\Gamma}{\Omega} w_t^s\right) \right\}, \tag{8.31}$$

where, the first term is the overall inter-deployment point dislocation distance, and the second term approximates the length of each active arm. The first term is scaled by $|N|$ because all the robots travel together from two successive deployment points. To more realistically account for the average distance covered by the *backbone robots* of each arm, the second term could be scaled by $\frac{w_t^s + 1}{2}$ (note that $\sum_{n=1}^{n} i = \frac{n\,(n+1)}{2}$); however, this operation is ignored to avoid the introduction of non-linearities. Eight new groups (four for the first time slot and four for the remaining ones) of constraints are added to correctly compute the $X$-$Y$ distance between two successive deployment points:

$$\bar{g}^{Xs} \geq g^{Xs} - \Theta_0^X, \qquad \forall s \in S_0, \tag{8.32}$$

$$\bar{g}^{Xs} \geq -g^{Xs} + \Theta_0^X, \qquad \forall s \in S_0, \tag{8.33}$$

$$\bar{g}^{Ys} \geq g^{Ys} - \Theta_0^Y, \qquad \forall s \in S_0, \tag{8.34}$$

$$\bar{g}^{Ys} \geq -g^{Ys} + \Theta_0^Y, \qquad \forall s \in S_0, \tag{8.35}$$

$$\bar{g}^{Xs} \geq g^{Xs} - g^{Xs-1}, \qquad \forall s \in \bar{S}, \tag{8.36}$$

$$\bar{g}^{Xs} \geq -g^{Xs} + g^{Xs-1}, \qquad \forall s \in \bar{S}, \tag{8.37}$$

$$\bar{g}^{Ys} \geq g^{Ys} - g^{Ys-1}, \qquad \forall s \in \bar{S}, \tag{8.38}$$

$$\bar{g}^{Ys} \geq -g^{Ys} + g^{Ys-1}, \qquad \forall s \in \bar{S}. \tag{8.39}$$

The optimal completion time $\Lambda$ computed by the TCTM formulation ( 8.9- 8.30) is protected by the following constraint:

$$\sum_{s \in S} z^s \leq \Lambda. \tag{8.40}$$

The domain of the new variables is expressed by:

$$\bar{g}^{Xs},\, \bar{g}^{Ys} \in \mathcal{R}, \qquad \forall s \in \bar{S}. \tag{8.41}$$

The remaining constraints are kept unchanged from the original TCTM formulation:

$$(8.10) - (8.30) \tag{8.42}$$

**Considerations on Decentralization** Providing an exact MILP formulation for the new swarm scheduling problem does not mean that the TSG must operate in a centralized fashion. In fact, once the exact problem has been correctly defined in terms of constraints, decision variables and objectives, it is up to each application manager to evaluate how to compute nearly optimal solutions within the desired time-limit. For instance, it would be possible

to adopt a centralized approach where a formation leader [23] directly solves the MILP model, as well as runs a centralized heuristic [87], to compute a TSG solution to be later distributed to the other swarm members. On the other side, each swarm member could be asked to heuristically compute a TSG solution and later leverage a consensus-based approach to negotiate the global task schedule. The importance of formulating and solving the exact MILP model for the TSG problem can be summarized in five main reasons:

1. Definition of the constraints and decision variables to be considered by heuristic approaches.
2. Computation of the lower bounds (optimal or sub-optimal depending of instance dimensions) to evaluate the performance of heuristic approaches.
3. Identification of the mathematical elements that can be leveraged to build more efficient math-programming tools (e.g., column generation, bender decomposition, etc.).
4. Analysis of the system behaviour through the optimal solutions of instances solved within the time-limit.
5. Development of approaches based on the MILP resolution for small- and mid-sized instances.

## 8.4 Experimental Set-up

To validate the RNC and TSG presented in the previous section, we set up experiments and simulations using the tools and parameters described in the following.

### 8.4.1 Robot Navigation Controller

The SECS algorithm exploited by the RNC was first implemented in an idealized scenario—in which we assumed no robot collisions nor communication interferences—using Python and Jupyter Notebook [6]. The parameters that we used for these simulations were the number of robots $N_p \in [10, 20, 30, 40, 60, 90]$, the number of tasks $T_p \in [1, 3, 4]$, the size of the squared arena in which the robot moves $D_p = 8\text{m}$, and the communication range between robots $r_p = 1\text{m}$.

After the initial results seemed to validate SECS, we moved to its implementation in a more realistic scenario with the aid of the multi-physics robot simulator ARGoS [133]. ARGoS can efficiently simulate large-scale swarms of robots of any kind and it model complex real-life interactions, including collisions, inertia, robots obstructing the sight, movement, and communication of other robots, etc. ARGoS supports robot controllers written in C++, however,

6. git@github.com:JacopoPan/ar-prominences-in-the-ideal-world.git

for our second implementation of SECS we chose to use Buzz, an internally developed and swarm-specific programming language [131].

Using ARGoS and Buzz, we performed simulations in which the number tasks $T_a$ was varied between 1, 3, and 5. For the number of robots $N_a$, we used swarms of size 3, 5, 10, 15, 20 and 25 (backbone robots, not including the master robot). The arena we used for the simulations was $D_a = 30$meters in length and width. The maximum linear speed of each robot ($\sim$ 10cm in diameter) was limited to 15cm/s. The communication range between robots $r_a$ was set at 3meters. We observe that, despite $D_p \neq D_a$ and $r_p \neq r_a$, the ratios $D_p/r_p$ and $D_a/r_a$ are comparable, meaning that the robots have similar room to move in the Python and ARGoS simulations. With regard to the Lennard-Jones potential parameters, we used $\epsilon = 10^6$, $\delta \in [90.0, 120.0, 240.0, 275.0]$, and exponents of 4 and 2 (see Figure 8.5). In our discussion, we explore how the choice of the $\delta$ parameter affected the ability of SECS to create prominences of different density (and how this echoed on their propagation time and reliability/link redundancy). All these implementations are available under the MIT license on GitHub [7].

### 8.4.2 Task Scheduling Controller

Both TCTM and TDSM were tested over 720 random instances of the problem. The experiments were carried out on machines equipped with an Intel(R) Core(TM) i7-3770 and 32 GB of RAM. We relied on AMPL as modelling language, while we used CPLEX 12.7 with `threads = 8` and `mipemphasis = 1` to solve the MILP formulations. For each instance, we considered a time-limit of 3 hours and solved, sequentially, the TCTM and the TDSM problems. For TDSM we used the maximum duration $\Lambda$ returned by TCTM. All instances were characterized by a square arena of side $L$, where:

$$L = r^X_{max} - r^X_{min} = r^Y_{max} - r^Y_{min}. \tag{8.43}$$

$L$ was determined as in [132], according to the communication density value $D$ and the robot cardinality $|N|$:

$$L = \sqrt{\frac{N\pi\Gamma^2}{D}} \tag{8.44}$$

In each instance, we considered a capability set $C$ of cardinality 2. We used $p_{Rob}$ and $p_{Tsk}$ in $[0, 1]$ to determine the probability for a robot of having a capability and that for a task of demanding a capability (same probability for each capability of the set). Furthermore, let

7. git@github.com:JacopoPan/ar-argos-buzz-simulations.git

$\beta_{max}$ and $\delta_{max}$ be the maximum number of robots that a task may require and the maximum allowed task duration. During the generation, $\beta$ and $\delta$ values were chosen according to a uniform distribution within the integer set delimited by 1 and the corresponding maximum value. The location of each task ($\left(\sigma_t^X, \sigma_t^Y\right)$), as well as the starting deployment point ($\left(\Theta_0^X, \Theta_0^Y\right)$) were chosen uniformly within the square arena. We considered no priority among the tasks ($\mu$ parameters all equal to 0). Both communication range $\Gamma$ and reliability value $\Omega$ were fixed to 1.

We summarize in Table 8.2 the instance classes used for the test campaign. Note that for each instance class (column `ID`), we generated `inst` instances for each of the following values of communication density $D$, i.e., 0.005, 0.01, 0.05, 0.1, 0.2 and 0.5 (corresponding to a decreasing arena width $L$), for a total of six times `inst` instances per class.

## 8.5 Experimental Results and Discussion

This section is dedicated the presentation of the results of our experimental evaluations for the RNC and TSG.

### 8.5.1 Robot Navigation Controller

The results of the Jupyter Notebook implementation (see Figure 8.6) have already been reviewed in Subsection 8.3.2 and persuaded us to move from that naive to a realistic multi-physics implementation. Looking at the results of the simulations in Figure 8.6, the most striking feature of SECS is how well it appears to scale from a mid-sized swarm scenario ($N = 15$) to a large swarm scenario ($N = 60$).

We could not be sure, however, that these results would have propagated into the experiments performed with ARGoS once the complex effects of robot collisions and communication interferences were added. These phenomena, in fact, can severely hinder the message-passing process down the structure of each prominence. This, in turn, can have unpredictable effects—e.g., on the calculation of the distance gradient or the potential contribution of a robot eclipsed by a closer neighbour—and alter the decisions taken by the RNC.

The results of experiments such as the one presented in Figure 8.7 ($N = 10$, $T = 1$), show that the algorithm is capable of overcoming these issues, as well as small collisions, and the loss of a few messages does not prevent the eventual insurgence of the desired behaviour. Figure 8.7(a) presents and example scenario in which $\delta = 90$ as a Lennard-Jones parameter makes the robot stick close together and create thicker prominences. These kind of prominences reach less further but are more reliable to robot failures (higher link redundancy). Figure 8.7(b)

displays the same experiment but with a $\delta$ parameter if 240. This suffices for the creation of much thinner prominences that can achieve tasks in more remote locations. In Figure 8.8, we provide the results of an experiment that demonstrates how SECS can drive a mid-sized swarm ($N = 16$) towards multiple tasks ($T = 3$) that are located at the same distance from the swarm centroid and have angular separation of 120$\circ$ from one another. The value of $\delta$ used in this test is 240 and the time to reach fully deployment is in the order of a few minutes ($\sim 180''$).

Table 8.1 summarizes the averages of the results of the experiments we conducted with Lennard-Jones potentials parametrized by $\delta = 120$ $\delta = 240$. The size of the swarm was varied from 3 to 20 robots. As we would expect, there is visible correlation both (i) between the number of robots in a prominence and the length of the prominence and (ii) between the value of $\delta$ and the length of the prominence (i.e., linear trends down the columns and between corresponding entries in the top and bottom part of the table). With regard to convergence and latency, we observe that SECS still scales well: more robots do not slow-down (but speed-up) the creation of short prominences, in the order of $\sim 10^1$ seconds. The full stretch of a prominence with $\sim 10^1$ robots requires $\sim 10^2$ seconds. With regard to fault tolerance, we report the densities of the prominences (in robots/meter) to show that even the longest prominences ($> 20$ meters have density $> 0.6\bar{6}$ that, with $r = 3$ means, on average, a double path on every communication link.

Finally, it is worth mentioning that two of the most crucial steps in the algorithm of SECS, i.e., the exchange of coordinate systems and the alignment maneuver, have already been successfully tested in a real-world robot simulation using Kheperas (the robots shown in Figure 8.1).

### 8.5.2 Task Scheduling Controller

We observe that TCTM and TDSM performance are correlated with the instance features. First of all, the total duration values $\Lambda$ reported in Table 8.3 show, as expected, that the higher the robot density, the lower the optimal execution time: task locations get closer, fewer robots are assigned to each arm, and more robots are available to complete additional tasks in the same time slot. Note that except for instance class 5, all tasks have a unitary completion time; thus, the total completion time is equal to the number of time slots used. We do not differentiate among TCTM and TDSM solutions because they showed the same $\Lambda$ values. Solution examples are presented in Figure 8.9 and Figure 8.10. Although this equivalence is guaranteed when TCTM returns the optimal solution ($\Lambda$ cannot be further improved), this is not the case for the instances that are solved by TCTM with an optimality

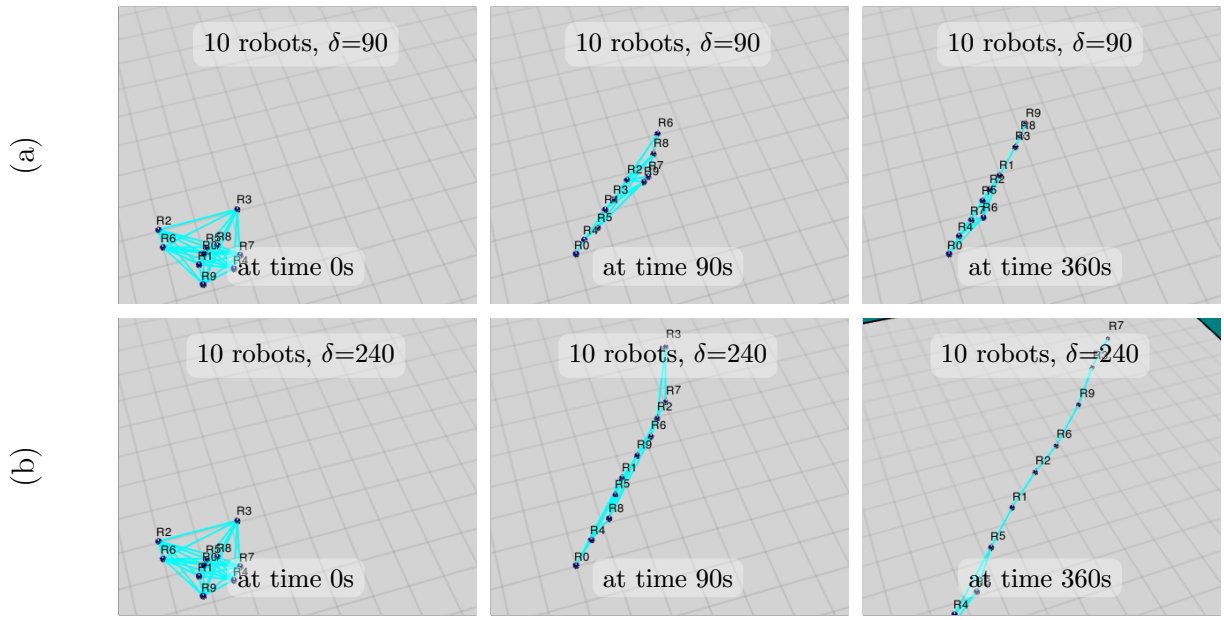



Figure 8.7 The RNC (implemented with Buzz programming language and tested in the ARGoS simulation environment) produces eruption of prominences with the same number of robots different lengths in relation to the value of $\delta$—90 in (a), 240 in (b)—used to parameterized the neighbour potentials.

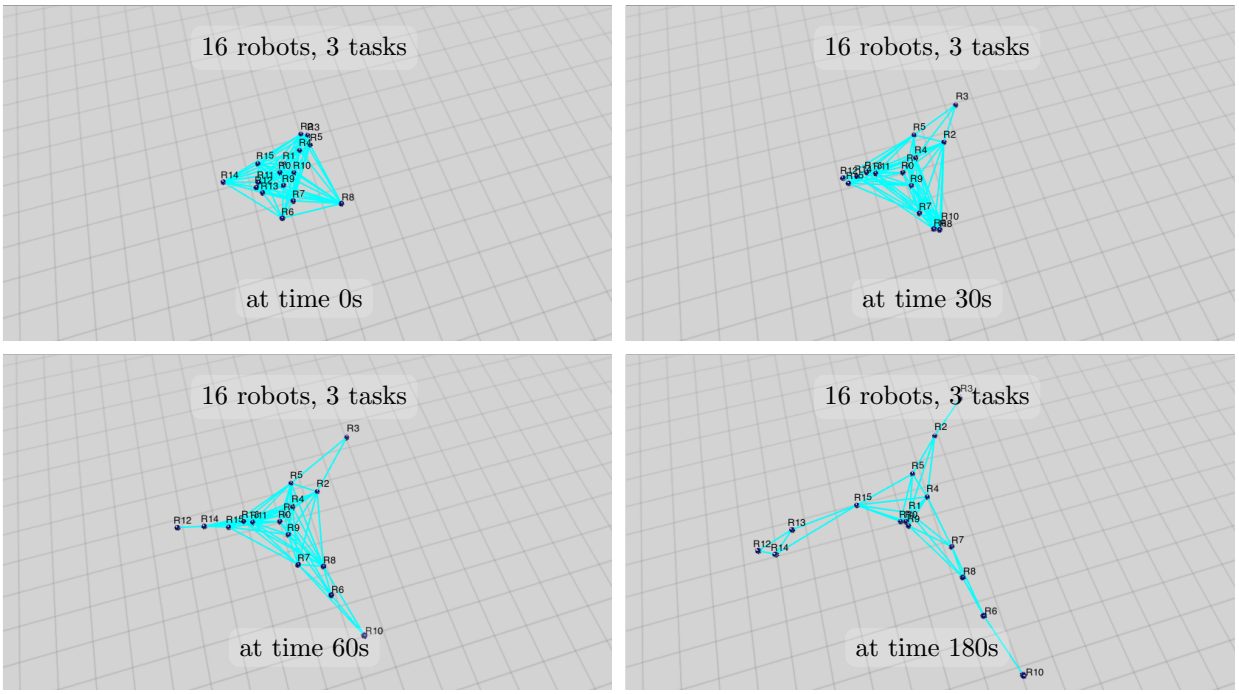



Figure 8.8 The RNC (implemented with Buzz programming language and tested in the ARGoS simulation environment) drives a swarm of 16 robots towards three different task directions from the swarm centroid.

Table 8.1 Average ARGoS/Buzz prominence eruption times.

| $\delta$ | Number of Robots | Latency (s) and Density (robot/m) $\rightarrow 2\sqrt{2}$m | $\rightarrow 3\sqrt{2}$m | $\rightarrow 10\sqrt{2}$m | $\rightarrow 15\sqrt{2}$m |
|---|---|---|---|---|---|
| 120 | 3 | $n/a$ | $n/a$ | $n/a$ | [< 1.89m] |
| | 5 | 81.1, 1.77 | $n/a$ | $n/a$ | [< 3.39m] |
| | 10 | 36.3, 3.54 | 69.8, 2.36 | $n/a$ | [< 7.31m] |
| | 15 | 37.9, 5.30 | 56.4, 3.54 | $n/a$ | [< 12.02m] |
| | 20 | 50.9, 7.07 | 63.2, 4.71 | 345.0, 1.41 | [< 18.38m] |
| | Number of Robots | Latency (s) and Density (robot/m) $\rightarrow 3\sqrt{2}$m | $\rightarrow 5\sqrt{2}$m | $\rightarrow 10\sqrt{2}$m | $\rightarrow 15\sqrt{2}$m |
| 240 | 3 | 69.8, 0.71 | $n/a$ | $n/a$ | [< 4.95m] |
| | 5 | 63.7, 1.18 | 195.1, 0.71 | $n/a$ | [< 7.78m] |
| | 10 | 57.6, 2.36 | 100.1, 1.41 | $n/a$ | [< 11.31m] |
| | 15 | 41.9, 3.54 | 74.2, 2.12 | 327.4, 1.06 | 449.8, 0.71 |
| | 20 | 43.3, 4.71 | 86.1, 2.83 | 237.2, 1.41 | 322.6, 0.94 |

gap greater than 0; in that situation, Constr. (8.40) would allow the TDSM formulation to further reduce the global completion time $\Lambda$.

The computation times are reported in Table 8.4. The two main insights are: (i) the instances get easier to be solved as the communication density $D$ grows, (ii) the TDSM formulation is computationally more expensive than the TCTM formulation. Higher $D$ means less time slots needed and many more equivalent solutions in terms of task scheduling; on the other side, with a more sparse scenario, the solver requires additional time to evaluate all the potential corner scheduling solutions that may further decrease the completion time. With TDSM, this phenomenon is less conspicuous (see column 0.2 in Table 8.4) because other solution components besides the task-time slot assignment, e.g., the specific deployment points, concur to determine the optimal solutions. The complexity related to the location of each deployment point and to the arm lengths is the factor that makes the TDSM formulation more computational expensive. However, this additional computation is often well allocated, as shown in Figure 8.11, TDSM optimal solutions drastically decrease the dislocation spaces that all the element of the swarm must cover between successive time slots. The same improvement is not observed in the case the arm costs (second term of Objective function 8.31) reported in Figure 8.12.

It is worth pointing out how the instance features may affect the computing times. It strongly stands out that heterogeneous task durations $\delta$ (class 5) make both TCTM and TDSM more computationally expensive, which results in the time-limit being reached in most of the

instances (the returned solutions have a gap from the best lower bound found at the moment of the time-limit expiration); heterogeneous task durations make the implicit bin-packing problem more complex (it is better to match together in the same time-slot all the longest tasks) and make the Linear Programming (LP) relaxation used to search for integer solutions and improve the lower bound significantly less efficient. As expected, computation times deteriorate by increasing up to 30 the number of tasks (combinatorial explosion). In that case, CPLEX reaches the time-limit for all the TDSM instances.

Finally, Table 8.5 report the gaps of CPLEX solutions from the best lower bound. Obviously, all the solutions returned before the time-limit threshold are optimal and have a 0% gap. For classes 1-2-3-4, the very small gaps observed for some values of $D$ (in the order of 1%) are due to one or two instances that reached the time-limit over the 30 in the instance set. In the case of classes 5 and 6, gaps are instead remarkable; however it is worth pointing out that a high gap may be both caused by a poor solution and a poor lower bound: in this case we are confident that we are falling in the second case. In fact, by looking at the logs of the instances solved at optimality, we remarked that quasi optimal solutions were found very soon during the elaboration, while most of the time was spent by the solver to improve the lower bound. Unfortunately, the lower bound improvement process is quite inefficient because of the so-called big-M constraints (Eq. 8.17- 8.19) that deteriorate the quality of the LP relaxation.

Table 8.2 Parameters for instance generation.

| ID | $\|N\|$ | $\|T\|$ | $\|S\|$ | $\beta_{max}$ | $\delta_{max}$ | $p_{Rob}$ | $p_{Tsk}$ | `inst` |
|---|---|---|---|---|---|---|---|---|
| 1 | 100 | 20 | 20 | 1 | 1 | 1 | 1 | 30 |
| 2 | 100 | 20 | 20 | 1 | 1 | 0.5 | 0.5 | 30 |
| 3 | 100 | 20 | 20 | 5 | 1 | 1 | 1 | 30 |
| 4 | 100 | 20 | 20 | 5 | 1 | 0.5 | 0.5 | 30 |
| 5 | 100 | 20 | 20 | 1 | 10 | 1 | 1 | 15 |
| 6 | 100 | 30 | 30 | 1 | 1 | 1 | 1 | 15 |

Table 8.3 Optimized duration values $\Lambda$.

| ID | 0.005 | 0.01 | 0.05 | 0.1 | 0.2 | 0.5 |
|---|---|---|---|---|---|---|
| 1 | 7,47 | 5,87 | 3,90 | 3,07 | 2,93 | 2,00 |
| 2 | 7,33 | 5,93 | 3,97 | 3,00 | 2,93 | 2,00 |
| 3 | 7,67 | 6,17 | 4,00 | 3,20 | 3,00 | 2,17 |
| 4 | 7,73 | 6,20 | 4,00 | 3,00 | 3,00 | 2,17 |
| 5 | 49,93 | 41,71 | 27,87 | 23,60 | 19,13 | 15,33 |
| 6 | 9,47 | 7,93 | 5,07 | 4,27 | 3,13 | 3,00 |

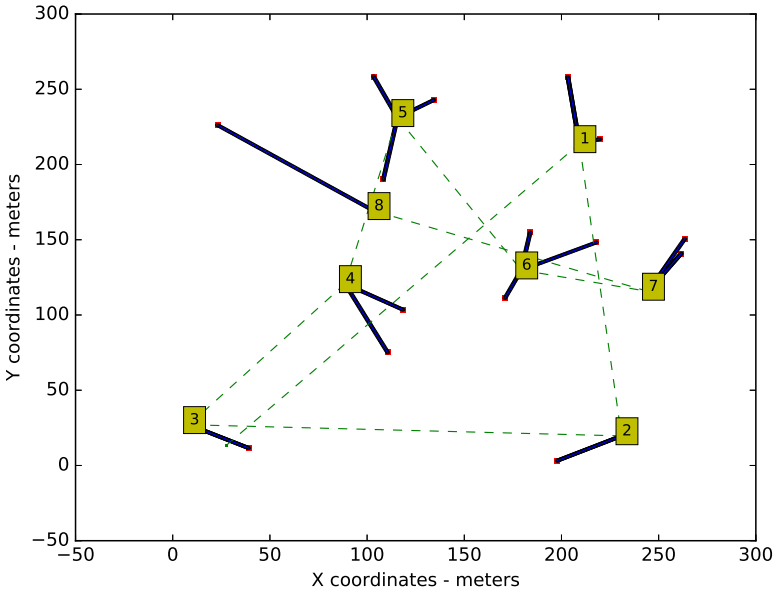



Figure 8.9 Example of TCTM solution (with 100 robots and 20 tasks) that only uses deployment points (SCs) associated to two or three tasks (i.e. two or three prominences).

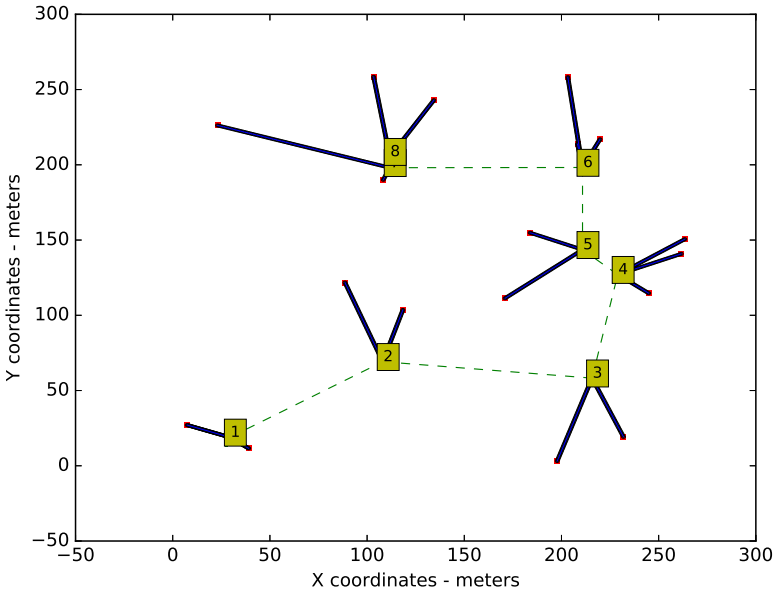



Figure 8.10 Example of TSDM solution (with 100 robots and 20 tasks) that uses deployment points (SCs) associated to two to four tasks (i.e. two to four prominences).

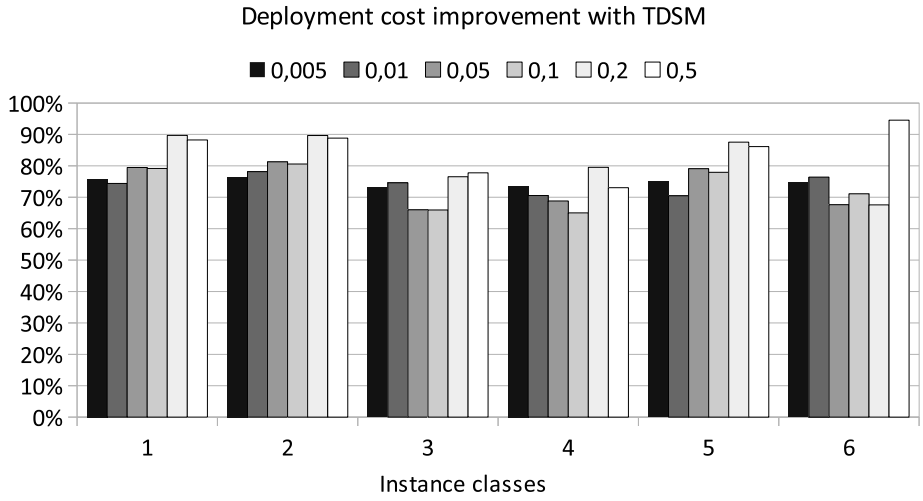



Figure 8.11 Analysis of the improvement achieved by TDSM w.r.t. to TCTM in terms of dislocation costs, i.e., $\sum_{s\in\bar{S}}\left(\bar{g}^{Xs}+\bar{g}^{Ys}\right)$.

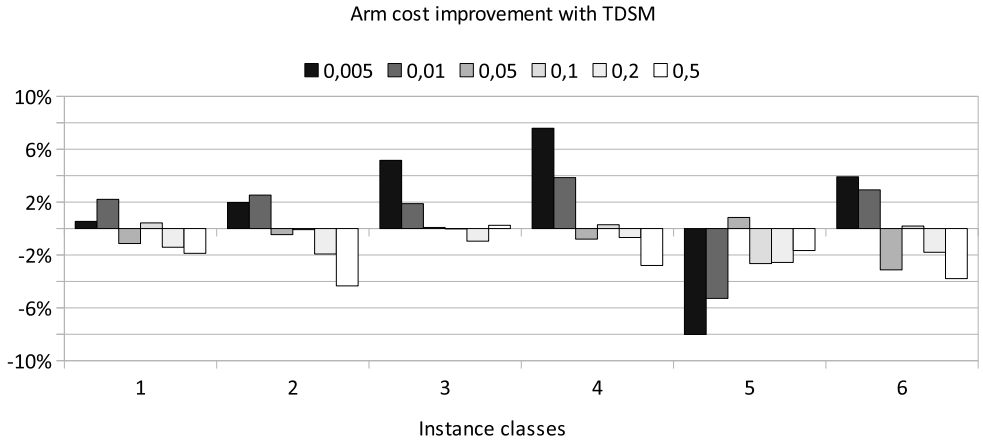



Figure 8.12 Analysis of the improvement achieved by TDSM w.r.t. to TCTM in terms of arm costs, i.e., $\sum_{s\in S,t\in T}\left(\frac{\Gamma}{\Omega}w_t^s\right)$.

## 8.6 Conclusions and Future Work

To summarize, in this work, we started from a critical analysis of the literature on the subject of distributed multi-robot connectivity assessment and maintenance to motivate our proposal of a hybrid methodology to address the problem of the spatial coverage of multiple tasks using a swarm of robots that preserve their global connectivity. Our approach has beed implemented *via* two layers: the fully distributed Robot Navigation Controller that

Table 8.4 Computational times.

| TCTM | | | | | | |
|---|---|---|---|---|---|---|
| ID | 0.005 | 0.01 | 0.05 | 0.1 | 0.2 | 0.5 |
| 1 | 908,1 | 206,4 | 15,3 | 7,0 | 4,9 | 0,8 |
| 2 | 210,7 | 62,8 | 3,8 | 3,6 | 2,8 | 0,7 |
| 3 | 1488,4 | 479,6 | 13,6 | 12,2 | 1,7 | 1,9 |
| 4 | 1362,1 | 202,2 | 27,9 | 44,8 | 2,1 | 3,3 |
| 5 | 8038,6 | 9626,5 | 6350,4 | 1006,1 | 134,1 | 9,0 |
| 6 | 8207,5 | 8530,3 | 9941,1 | 3340,6 | 2111,6 | 95,7 |
| TDSM | | | | | | |
| ID | 0.005 | 0.01 | 0.05 | 0.1 | 0.2 | 0.5 |
| 1 | 2455,4 | 1657,7 | 811,8 | 488,0 | 2254,3 | 44,3 |
| 2 | 1036,9 | 1111,5 | 530,8 | 174,2 | 2332,1 | 50,5 |
| 3 | 2817,4 | 1998,0 | 524,9 | 1338,8 | 772,1 | 1049,3 |
| 4 | 4890,9 | 2200,1 | 781,1 | 2149,4 | 1249,0 | 1453,8 |
| 5 | 10800,0 | 10800,0 | 10800,0 | 10094,7 | 5719,1 | 31,6 |
| 6 | 10800,0 | 10800,0 | 10800,0 | 10800,0 | 10800,0 | 10800,0 |

Table 8.5 Optimality gaps w.r.t. the best lower bound computed by the solver.

| TCTM | | | | | | |
|---|---|---|---|---|---|---|
| ID | 0.005 | 0.01 | 0.05 | 0.1 | 0.2 | 0.5 |
| 1 | 0,7% | 0,0% | 0,0% | 0,0% | 0,4% | 0,0% |
| 2 | 0,0% | 0,0% | 0,0% | 0,0% | 0,0% | 0,0% |
| 3 | 2,0% | 0,0% | 0,0% | 0,0% | 0,0% | 0,0% |
| 4 | 0,4% | 0,0% | 0,0% | 0,0% | 0,0% | 0,0% |
| 5 | 43,9% | 31,3% | 5,8% | 0,0% | 0,0% | 0,0% |
| 6 | 31,6% | 33,6% | 14,2% | 12,3% | 3,3% | 0,0% |
| TDSM | | | | | | |
| ID | 0.005 | 0.01 | 0.05 | 0.1 | 0.2 | 0.5 |
| 1 | 0,5% | 1,7% | 0,0% | 0,0% | 0,9% | 0,0% |
| 2 | 0,0% | 0,0% | 0,0% | 0,0% | 0,2% | 0,0% |
| 3 | 0,8% | 0,0% | 0,0% | 2,1% | 0,0% | 1,6% |
| 4 | 0,8% | 0,6% | 0,0% | 0,4% | 0,0% | 2,6% |
| 5 | 69,2% | 77,8% | 95,8% | 83,6% | 30,8% | 0,0% |
| 6 | 70,9% | 71,2% | 81,9% | 76,2% | 59,6% | 43,1% |

produces the collective behaviours—flocking, rendez-vous, and eruption of prominences—and guarantees the connectivity of the swarm; and a global planner (the Task Schedule Generator) that approximates the best strategy for the RNC (with minimal computational load) for two possible optimization metrics—Total Completion Time Minimization and Total Dislocation Space Minimization.

The entirety of the source code of our original designs and literature re-implementations is being made available on GitHub. In the results section, we presented simulations of the RNC performed in a multi-physics environment that shows how our distributed control strategy can build long multi-robot structures, in multiple directions, without breaking the swarms connectivity. The TSC experimental results also showed how it is possible to find near-optimal coordination strategies for the RNC with affordable computational times. In the future developments of this research we plan on extending the RNC with additional complex behaviours (e.g., the ability to build slime mold and fractal-like formations) and to expand the discussion on autonomous robot control with the introduction of a new interface layer for human-robot coordination.

# CHAPTER 9 GENERAL DISCUSSION

"To him who looks upon the world rationally,
the world in its turn presents a rational aspect. The relation is mutual."
"Wer die Welt vernünftig ansieht, den sieht sie auch vernünftig an, beides ist in Wechselbestimmung."
Georg Wilhelm Friedrich Hegel, *Vorlesungen über die Philosophie der Geschichte*, 1837

"Thoughts without content are empty, intuitions without concepts are blind."
"Gedanken ohne Inhalt sind leer, Anschauungen ohne Begriffe sind blind."
Immanuel Kant, *Kritik der reinen Vernunft*, 1781/1787

This chapter is dedicated to the discussion of the results brought by the research presented in the previous Chapters from 4 to 8. The analysis is divided into four sections to cover the same four areas—hardware design, adaptive systems, resilience, and swarm robotics—used to split the review of the literature in Chapter 2. The aim is to recapitulate the main findings as well as to underline inter-disciplinary connections, their impact, and their potential to alter the landscape offered by the exiting literature. At the end of chapter, Table 9.1 summarizes this analysis.

## 9.1 On Hardware-software Co-design and Optimization

The aerospace sector has a special interest in computing systems with *ad hoc* hardware designs (and methodologies to create them): for example, to realize MPSoCs that meet the specific real-time requirement of a satellite's software architecture using only a subset of the available components (e.g., those that are radiation hardened and/or have low-power consumption). The results in this area came primarily from the work carried out in WP1 and, in part, WP3. The heart of the research developed in WP1 was published in [116] and presented in Chapter 4.

In Chapter 1, the problem statement highlighted the "uncertainty surrounding the choice of the most appropriate algorithms [..] for the automated optimization of an embedded system design" and the research objectives included the desire to "conceive a formal method to classify and compare the existing methodologies for the design-space exploration of an embedded system." Chapter 4 proposes a 4-class taxonomy for design-space exploration algorithms—(i) heuristics and pseudo-random searches; (ii) genetic and evolutionary algorithms; (ii) statistical methods without domain knowledge; and (iv) statistical methods with domain knowledge.

The surveys in the literature, prior to this work, typically focussed on methodologies from only one of these areas at a time. For example, [181] and [31] are reviews of evolutionary and genetic algorithms for multi-objective optimization problems, respectively. The experimental results in Chapter 4 point out that, for a few of the evaluation metrics (convergence, ADRS), certain inter- and intra-class trends exist—a discovery that can improve how one selects a strategy to automate the design of embedded systems. The core results in Section 4.6.3 show that there is little variance in metrics such as "number of Pareto points", "non-uniformity", or "concentration" across all the algorithms under test. However, there is clear disparity in: (i) the rates of convergence—the methods in classes 3 and 4 require to evaluate fewer points; and (ii) ADRS (error at convergence)—the methods in classes 2, 3, and 4 typically fare much better than simpler heuristics. A few corollary results about DSE uncovered by Chapter 4 are: (i) the importance of choosing representative benchmark applications to obtain results that generalize; and (ii) the challenges of choosing appropriate quality metrics for Pareto sets [109] when multi-objective scalarizing functions [104] are not available.

The analysis in Section 4.7 suggests that the selection of the best DSE algorithm for the optimization of a specific application is often a difficult task. In fact, no algorithm clearly dominates all others. Using the results from Section 4.6, a set of guidelines is derived. These indications map (i) the size of a design space and (ii) the cost of each evaluation into a short list of recommended approaches. Higher evaluation costs (complex or difficult to emulate systems) make algorithms that require a small number of simulations more appealing—even if they have longer configuration times. Conversely, if evaluations are cheap, one would prefer algorithms with effortless setup, even at the cost of a higher number of simulations. A universally-agreed and formal method to quantify the setup effort of a DSE algorithm does not yet exist. Chapter 4 introduces a qualitative-quantitative approach, taking into account the user feedback and the number of parameters in each algorithm. A novel, maybe counter-intuitive finding is that, when the problem is of moderate complexity (i.e., cheap evaluations and mid-sized design-spaces), the best choice is to rely on non-evolutionary heuristics such as Multi-Objective Multiple Start Local Search (MOMSLS) and Adaptive Windows Pareto Random Search (APRS). For highly complicated design-spaces, methods that exploit domain knowledge and/or statistical models are best. The hybrid Ishibuchi-Murata MO Genetic Local Search (IMMOGLS) approach is the only heuristic/GA-based method with comparable performance in this predicament (Alouani *et al.* [5] and Mediouni *et al.* [101] exploit this dissertation's result). The discovery that the performance of DSE optimization depends on certain high-level aspects of the design-space implies that: (i) a designer can now efficiently choose an appropriate exploration algorithm; and (ii) the developer of re-usable optimization frameworks shall include multiple exploration strategies within their products, to cope with

different types of design spaces.

The results of Chapter 4 also form the bedrock of a chapter with title "Optimization Strategies in Design Space Exploration" in the forthcoming "Handbook of Hardware/Software Codesign" edited by Soonhoi Ha and Jürgen Teich. Additional findings on how to direct the design phases of computer hardware also came from [115] and [117]: the work in Chapter 5 can be used to define an appropriate (yet frugal) level of redundancy to meet a given lifetime requirement; the work in Chapter 6 suggests the appropriate number of PEs in a real-time multiprocessor to achieve a desired level of fault tolerance. Combined, these two results can pave the way to cheaper aerospace on-board computers exploiting the most recent general-purpose hardware while still meeting their more stringent requirements.

## 9.2 On Adaptive Hardware and Software

Criticality and limited-access are the two aspects of the space environment that make adaptiveness so important for its computing devices. An astronaut cannot shop for a new computer if the performance of a life-critical instrument degrades or the computational requirements of a scientific application suddenly change. The challenges exposed by Chapter 1 included the "inefficiencies—in terms of computing resources utilization, power consumption, and fault tolerance—of traditional redundancy schemes used in aerospace" and motivated the desire to "discover new, non-obvious relationships between performance metrics such as energy consumption, [and] real-time execution" in order to "implement adaptive fault tolerance". The results in this area were produced by WP2 and WP3. The work realized in WP2 and 3 was presented in Chapters 5 [115] and 6 [117]. Additional material from WP2 is also recapitulated in Appendix B [167].

In low Earth orbit or during interplanetary travel, the space environment can alter the performance and fault rates of the computing systems that traverse it. The reason for this is the high radiation comprising of solar wind, gamma, and cosmic rays. In CMOS circuits, the consequences are premature failures—due to the cumulative effects of the total ionizing dose—and transient errors—e.g., single event upsets [128]. As these environmental conditions continuously change—even from one portion to another of a satellite's orbit—adaptive hardware and software become a necessity for the simplification of aerospace computing systems and the containment their design and manufacturing costs.

Realizing adaptive hardware and software for aerospace applications requires two main steps: (i) modelling the dynamic radiation environment of a spacecraft; and (ii) devising strategies to adjust to it. The research in Chapter 5 [115] tackles the first problem with the aid of

probability theory. Yet, it has to be fed with minimal domain knowledge (i.e., the average frequency of particle impact strikes and the components' MTTF) for parameterization. The software framework from Appendix B [167]—exploiting the phenomenological model from [164]—can be used to this end. The work in [167] also automates the process that—from the two-line element set of a satellite—directly injects bit-flips into FPGA fabric, as if the device was on-board the spacecraft. The joint use of probability theory, physical models, and error injection can greatly accelerate the development of aerospace computing systems, as it provides an efficient test bench for sensitivity analysis and experimental pre-flight validation. *PolyOrbite*'s CSA-backed nano-satellite ORU-S (see Appendix A) and an upcoming MIST Lab's project on the RISC-V architecture will be among the first users of these tools.

Chapters 5 and 6 [115, 117] both outline adaptive strategies for resource management. The adaptiveness in [115] is based on the on-line estimation of "beliefs"—regarding the occurrence of transient and permanents faults—to migrate computation from one resource to another and to optimize the use and lifetime of a system. This approach (D-HMMs) breaks with traditional methodologies that make decisions based on thresholds fixed at design time. Even the previous mechanisms exploiting adaptive thresholds (e.g., exponential backoff) lack the statistically grounded semantic of the proposed methodology. Figures 5.5 and 5.6 compare the results of the proposed approach against a simpler probabilistic method and a rule-based system. The scenarios are those of a 10-resource system operating in a low Earth orbit and a highly elliptical orbit, respectively. While the rule-based system fails to achieve the desired lifetime, D-HMMs outperform the other probabilistic methodology. This can be explained by the better job, done by D-HMMs, capturing the cumulative effects of ionizing radiation on electronic devices. The application of these results has the potential to better exploit the high levels of redundancy in large spacecraft as well as to improve the lifetime of small satellites: a drastic change with respect to current designs—expensive and based on hardening. For example, NASA's US$1B Juno mission—currently orbiting through the intense radiation belts of Jupiter—only hosts a BAE Systems single board computer and 384 MBytes of memory within a 180 kilograms titanium vault.

Adaptiveness can also be implemented by analytically modelling hardware, software, and fault occurrences to uncover their interdependencies. Once a formal relation between two (or more) performance metrics is identified, one of these measures can be tuned *via* design or run-time choices to make the whole system adjust. In [117], this is done by modelling the effects of dynamic voltage and frequency scaling on both (i) the probability of unmasked transient errors (i.e., probabilistic fault tolerance) and (ii) energy consumption in the context of a real-time multiprocessor system. Figure 6.7 presents the feasible points in the utilization-space of a dual-core system, revealing those that exhibit ideal fault tolerance or energy

consumption (while still meeting the real-time requirements). The methodology is especially useful to determine the feasibility of integrating modern multiprocessor systems even on small spacecraft with limited power budgets.

Chapters 7 includes an application scenario revolving around a nano-satellite's multiprocessor computing system (Section 7.7). The developed theory on resilience is used to assess the need for dynamic software-to-hardware re-mapping so that the spacecraft can perform its mission through changing operating and environmental conditions that were unknown *a priori*.

All of these advancements come at a pivotal time for the space industry. Private companies and commercial applications appear destined to supplant traditional government-backed space programs. Between 2005 and 2012, private investments in space companies were estimated by the Tauri Group at US$12B (a level support that is expected to grow as the new industry develops) with venture capital and acquisitions accounting for the lion's share. In this expanding market, efficiency and profitability will be in high demand by new and old players desiring to consolidate their shares. In turn, this need will create the ideal context for the development and exploitation of adapting computing methodologies to cut costs while preserving (or improving) reliability—as the contributions presented here.

## 9.3 On the Resilience of Complex Systems

Resilience is a property of great interest for the study of many complex structures—including aerospace computing systems and autonomous multi-robot systems. In space, for example, a multiprocessor computer should be able to continue operating after the failure of one of its PEs and to adapt to still meet its real-time constraints. Similarly, an autonomous multi-robot team should seamlessly re-organize to cope with the failure of individual robots. The results presented in this area originated from the research conducted in WP4 (and, in part, the modelling in WP2, 3 and 5). The body of the work on resilience from WP4 was detailed in Chapter 7 and it is currently under review.

The problem definition in Chapter 1 identified the lacks of "a formal definition of what a resilient system is and what resilience entails" and of "formal models and methodologies to assess or quantify resilience". The research effort produced in this field, and presented in Chapter 7, aims at mitigating these inadequacies by modelling "aerospace computing systems through an intelligent framework that allows to answer (probabilistic) inference queries—in particular with regard to the system's resilience". Moreover, the contributions in Chapters 1, 5, and Appendix B are all devoted to the definition of "rigorous and realistic tools to model those aspects of the space environment that affect the operations of a computing system".

Resilience was initially introduced in the context of ecology [67] but it is now recognized as a distinctive trait of many social and economical systems. In Chapter 7, a formal definition of resilience—introduced originally in the context of constraint-based and dynamic systems by Schwind *et al.* [148]—is adapted to the timed probabilistic framework of hidden Markov models. To do so, HMMs are extended with cost functions into c-HMMs. Then, it is shown that the elementary queries required to assess the probability of long-lasting properties in this framework—including those comprised by resilience—can be answered by an exact algorithm whose complexity is linear with respect to the time horizon.

The experimental results in Chapter 7 show that the proposed approach to compute the probabilities of individual trajectories is, indeed, several orders of magnitude faster than expanding the full conditional joint probability distribution (Figure 7.7). Furthermore, four application scenarios (Section 7.7) demonstrate that the introduction of partial observability in the model can lead to insights about resilience that would be, otherwise, concealed—e.g., the link between the $q$ extra resources required to probabilistically ensure $\langle p, q \rangle$-recoverability and the tightness of the deadline associated to the depletion of $q$.

Nonetheless, we underline that an additional computational challenge introduced by long-lasting properties is the potential exponential growth of the number of state trajectories that satisfy a certain (resilient or not) property. This extra-layer of complexity cannot really be circumvented (but it could be mitigated by an approximate method). Without property-specific considerations, it is not possible to know *a priori* the number of trajectories to evaluate. For the "resistance" property (see Chapter 7), the proof of this claim is based on the observation that the probability of a sequence of states meeting instantaneous conditions each (e.g., $P(S_1 \leq l \wedge S_2 \leq l | o_1, o_2)$) cannot be factorized into a series of conditionally independent factors (e.g., $P(S_1 \leq l | o_1, o_2) \cdot P(S_2 \leq l | o_1, o_2)$).

The research presented in Chapters 5 [115] and 6 [117] (see also Appendices B and C with regard to the work in [167, 27]) adds to the study of resilience with two aspects that are peculiar to critical and aerospace computing systems: (i) the modelling of the natural environment that induces errors into CMOS electronics; and (ii) design strategies to assess and circumvent these errors. The work in [115] and [167] is an orthogonal contribution providing a probabilistic model of the space radiation environment for the development of FPGA-based computing systems (one of the first of its kind). The study in [27] is an investigation of how the methods in [115] affect the probabilistic timing analysis of a real-time system. Yet, resilience—a lot like intelligence or consciousness—is an elusive concept. Even in the narrow field of swarm robotics, different formulations are been proposed [146]. As of today, the only universally-agreed notion is that resilience is an inter-disciplinary subject. Understanding

and controlling resilience will benefit a myriad of different fields—from ecology to robotics, passing by economics and networking. The hope of this research is to foster the conversation around it and to contribute establishing it as a property of major interest for the designers implementing the next generation of computer and robotic systems.

## 9.4 On Multi-agent Systems and Swarm Robotics

Partially autonomous robots already wander on the surface of Mars [9] and—thanks to the recent advances in commercial space exploration and artificial intelligence—it is easy to imagine that this lineup will only grow in number and autonomous capabilities over the next few years (the upcoming NASA Jet Propulsion Laboratory's lander, InSight, is scheduled for launch in May 2018). On Mars, for example, a swarm of quad-rotors able to fly in thin air could be used to quickly deploy a communication infrastructure connecting distant astronaut teams. In this dissertation, the investigation of multi-robot applications was conducted through WP5. The results were collected into an article—presented in Chapter 8 and currently under review. Among the chapter's contributions, there is the implementation of "an adaptive and distributed algorithm that enforces connectivity in a multi-robot system" to address the lack of "a distributed and resilient methodology to preserve network connectivity in a multi-robot system exploring an unknown environment", as identified in Chapter 1.

Section 8.2 reviews previous research works dealing with the detection of topology changes and the assessment/control of network connectivity in swarm robotics. Most of them, however, were originally only validated through abstract models or in small scenarios with few robots. The reimplementation of these approaches allows to highlight the weaknesses, if any, that can curb their performance in real-world applications (Figures 8.2 and 8.3). In particular, the findings in Chapter 8 are as follows: (i) for the assessment of connectivity using spectral graph theory, power-iteration methods [17, 37] yield convergence times that are often too slow for practical usage; (ii) for the same goal, the performance of wave equation-based methods [145] degrades severely with noisy communication (the lesson here is that collisions and packet drop should not be overlooked); (iii) for the control of connectivity, methodologies based on mimicry and heuristics [89] can be vulnerable to overfitting; (iv) finally, one should observe that the detection of topology changes [18] is a useful clue for the assessment of connectivity but it can return a large number of false positives in rapidly moving swarms. The main engineering insight of these findings is that most theoretical frameworks (e.g., spectral graph theory) that provide solid foundations to the study of connectivity still require tweaking for practical usability.

The fundamental intuition behind the work in Chapter 8 is that—to pursue multiple objec-

tives whose positions are initially unknown—a swarm that wants to maintain connectivity should expand from its center towards the tasks, rather than being pulled from its extremities. As the information from the boundaries of the swam propagates more slowly, in fact, approaches such as [89] entail a higher risk of disconnection. The proposed approach, instead, does not stretch beyond its connectivity limits because it only propels the robots into elongated prominences using contributions from Lennard-Jones potentials that progressively nullify (with the growing distance from the origin, see Figure 8.7). A gradient is also run along the prominences to allow individual robots to learn about their estimated distance from the origin and improve the convergence time of the methodology. However, there is no pre-established ordering for the robots in a prominence, nor any fixed ordering is determined at runtime. This design choice can partially deteriorate the convergence time but greatly increases the resilience and fault tolerance of the swarm. In fact, if a robot fails, the team seamlessly re-organize without the need for any specific diagnostic and recovery routine.

Figure 8.8 demonstrates that the proposed methodology can drive a 16-robot swarm towards three targets (each five meters apart from the swarm's deployment point) in less than three minutes. The experimental results also highlight the correlation between (i) the number of robots in a prominence and the length of the prominence and (ii) the value of $\delta$ [1] and the length of the prominence. Table 8.1 shows that the approach scales extremely well, a valuable quality for swarm robotics. Adding more robots does not slow-down the creation of prominences and the full stretch of a prominence with $\sim 10^1$ robots requires $\sim 10^2$ seconds. With regard to the performance of the overall task scheduling policy, Table 8.3 shows, as expected, that the higher the robot density, the lower the optimal execution time. Table 8.4 provides two main insights: (i) the task scheduling problem becomes easier as the communication density grows and (ii) the TDSM formulation is computationally more expensive than the TCTM formulation.

Furthermore, the inference framework in Chapter 7 can be used orthogonally to assess *a posteriori* the performance of the resilient/robust methodologies proposed in Chapter 8, by Saldaña *et al.* [146], or Soleymani *et al.* [157]. The developed methodology has numerous applications but two, in particular, carry the greatest potential impact: (i) the reuse of this technology for more effective disaster response and (ii) the realization of better autonomous robotic exploration systems (for space or Earth). Fitly, the first two partners for which will develop applications based on this approach are: (a) HumanITas Solutions—a Canadian start-up providing technology services for humanitarian response and disaster relief—and (b) the European Space Agency.

1. One of the four parameters in the Lennard-Jones potential function (see Equation 8.5).

Table 9.1 Summary of the dissertation's main contributions.

| Ch. | Ref. | WP | Contribution | Significance/Potential Impact |
|---|---|---|---|---|
| 4 | [116] | 1 | A survey of the most popular DSE algorithms for multi-objective optimization of embedded systems. | Providing researchers and designers with an updated and comprehensive overview of the field. |
| | | | A novel four-class taxonomy to partition the existing research on DSE algorithms. | Helping to understand and predict the performance of a newly introduced DSE algorithm. |
| | | | Quantitative comparison and qualitative guidelines for the choice of the best DSE algorithms. | Facilitating/accelerating the DSE choices of designers. That is, better and/or cheaper final products. |
| 5 | [115] | 2, 3 | The joint modelling of the transient and permanent errors due to space radiation. | Unveiling design pitfalls in systems that accounted separately for these correlated phenomena. |
| | | | A strategy to efficiently use the slack/excess resources on redundant space computing system. | Improving the lifetime of space computing systems (alternatively, reducing their cost and size). |
| 6 | [117] | 3 | A fault tolerance/power consumption trade-off for homogeneous real-time multiprocessor systems. | Enabling the use of modern multiprocessors on-board critical aerospace computing systems. |
| 7 | – | 4 | A formal definition of probabilistic resilience in a timed and partially observable framework. | Enabling the creation of "resilient by design" computing and robotic systems. |
| | | | The study of the complexity of the exact inference steps required to probabilistically check resilience. | Allowing the implementation of the above designs in the context of critical, real-time applications. |
| 8 | – | 5 | A distributed controller based on potential forces to maintain the connectivity of a robot swarm. | Leveraging many (but cheap) robots to autonomously and reliably explore new environments. |
| | | | A formal definition of the task coverage problem for robot swarms and 2 metrics for its optimization. | Evaluating and comparing the performance of different exploration strategies for robot swarms. |

# CHAPTER 10 CONCLUSIONS

> "The more we learn about the world, and the deeper our learning, the more conscious, specific, and articulate will be our knowledge of what we do not know, our knowledge of our ignorance. For this, indeed, is the main source of our ignorance—the fact that our knowledge can be only finite, while our ignorance must necessarily be infinite."
>
> Sir Karl Raimund Popper, *Conjectures and Refutations: The Growth of Scientific Knowledge*, 1963

> "Perfecting oneself is as much unlearning as it is learning."
>
> Edsger Wybe Dijkstra, *Introducing a Course on Mathematical Methodology*, 1986

This research started with the aim to study adaptive computing system for aerospace and to define formal methodologies for their design and implementation. The six research objectives identified in Chapter 1 were accomplished through the five work packages presented in Chapter 3 and the research articles included in the body of this dissertation (Chapters 4 to 8, see also Figure 3.1). In particular, objective 1—discovering the best automated approaches for hardware optimization—was addressed in WP1 (Chapter 4); objective 2—modelling the effects of space radiation—in WP 2 (Chapter 5); objective 3—discovering novel performance trade-offs—in WP3 (Chapter 6); objective 4—developing an *ad hoc* framework for probabilistic inference—in WP3 and WP4 (Chapters 5 and 7); objective 5—use the same framework to study fault tolerance, resilience, and connectivity—in WP3 and WP4 (Chapters 5, 7, and 8); and finally, objective 6—implement an adaptive application in swarm robotics—in WP5 (Chapter 8).

The previous chapter discussed the impact of these results in the fields of hardware optimization, adaptive computing, probabilistic resilience, and swarm robotics. The scientific and engineering relevance of these contributions includes: 1. a set of guidelines to help computer hardware designers in the choice of the best automation tools; 2. a methodology to enhance reliability and resource utilization in small spacecraft designs; 3. a formal definition of resilience—applicable to multiple domains—and a framework for its assessment; and 4. an autonomous robotic framework to simplify disaster response and advance space exploration.

This final chapter summarizes a few of the lessons that were learned throughout this doctoral research and the recommendations that the author has for anyone who might be interest in implementing, reproducing, or further extend the investigation of adaptive computing systems for aerospace. Furthermore, Sections 10.2 and 10.3 list some of the questions that remain

open for investigation, and the prospective future work, respectively.

## 10.1 Lessons Learned and Recommendations

The research work presented in this dissertation developed over approximately five years and two continents. Of course, besides the bare numerical results, it came with a number of valuable lesson learned. These include:

1. Probability theory is a powerful tool to cope with the unknowns—e.g., defective sensors and unpredictable errors—as demonstrated by [115, 121].
2. "Fear the overfitting", as any machine learning practitioner would say. Overloading a model with excessive detail can revamp your results but make them impossible to reproduce, as it happened to [89] in [119].
3. Correlation does not imply causation nor causation implies correlation. Correlation only captures linear trends. Two random variables could be one function of the other and still show a null correlation if the function is symmetric with respect to the y-axis.
4. Problems that look simple on the surface and can (apparently) be described in just a few words might reveal themselves as extremely hard, even to formally define.
   - It was somehow counterintuitive to discover in [121] that the assessment of an instantaneous property—i.e., resistance—incurs, in the general case, in an unavoidable exponential growth in complexity with respect to considered time horizon.
   - Assessing network connectivity [17], preserving it, maintaining it while performing other spatial tasks [119], and detecting topology changes [18] are interconnected but all fundamentally different problems in swarm robotics.

The researcher who is interested in capitalizing on, replicating, or further developing the methodologies presented in this thesis, might also be interested in reading the following general recommendations:

1. Always formalize the research problem through a synthetic framework first. Introduce a novel one, if necessary. Then, investigate the theoretical and algorithmic complexity of finding the optimal solution within this framework.
2. Start the design process having in mind the desired properties that a system should manifest (e.g., resilience) and the constraints that it should meet (e.g., connectivity).
3. Any modern fault-tolerant scheme should reflect—and adjust to—all the available knowledge about the incidence of errors in the system under study.
4. In a multi-layer navigation controller, the connectivity of a robot swarm should be implemented at its lowest level and never overridden.

5. Welcome side projects: they often offer new angles and perspectives on your research as well as opportunities to sharpen your less mature skills.

## 10.2 Open Questions

As a research advances and the understanding deepens, one often discovers that the number of new questions that surface is just as large as that of those that get answered. This, after all, should not discourage us: new questions are *per se* new bits of knowledge. Two questions that appeared—but have not yet been answered—in the most recent phases of the investigation in this dissertation are as follows:

1. The concise but generic definition of resilience presented in [121] applies to a large family of stochastic models. Yet, we certainly do not have the arrogance to claim of having extinguished the debate: when or whether the scientific community will agree on how to quantify this property is hard to tell.
2. The minimum Steiner tree (for graphs) is the graph connecting a given set of vertices (and an arbitrary number of additional ones) whose set of arcs has the shortest overall length. Its decision problem is NP-complete [81] but it is often reasonably approximated by centralized approaches. When it comes to distributed approaches, however, is there a suitable heuristic to approximate a solution for this problem? Furthermore, would it be outperformed by another heuristic that does not seek the same optimality guarantees of a Steiner tree but can be more efficiently distributed?

## 10.3 Future Work

With regard to future work, the natural next steps of this research are the amelioration and further development of the contributions currently under review, that is, those presented in Chapters 7 and 8.

The work submitted in [121] mainly focussed on the theoretical aspects of resilience, that is, its formal definition and the algorithmic complexity of its probabilistic inference. A purely experimental extension of the work—to validate its everyday usability—would certainly help to clarify its potential impact. The work submitted in [119] culminated with the validation of the proposed methodology in a realistic but simulated multi-robot environment. The implementation on physical devices (e.g., K-team's Khepera IV robots or DJI's Matrice 100 drones) is an engineering effort that would immediately boost the marketability of the research. Promisingly, Buzz implementations have been transferred into real-world controller

before and with only modest adjustments.

### 10.3.1 Symbiotic Human and Multi-Robot Planetary Exploration

Furthermore, a proposal—loosely inspired by the work presented in Chapter 8 and [119]—has been preliminarily approved in the context of the European Space Agency's Networking/Partnering Initiative [1] The project enriches the research conducted so far with two new crucial facets: (i) the introduction of humans in the control loop; and (ii) the validation in the extreme environment represented by natural caves.

The project aims at developing the software infrastructure needed for one or more humans and a swarm of robots to collaborate in the exploration and mapping of planetary environments such as caves or lava tubes. As the robots explore, they shall dispose themselves so that network connectivity is guaranteed across the whole swarm—including the human(s). The overall goal is to increase the performance as well as the safety of the humans involved in the exploration. The specific objectives of the project are:

1. To further advance the development of algorithms for the self-organization of multi-robot system targeted towards (i) network maintenance and (ii) mapping of hazardous, unfamiliar environments.
2. To define new collaboration protocols between humans and robots for the exploration of unknown environments.
3. To create a low cognitive load interface for the control of a multi-robot system.

To do so, several incremental contributions are required: (i) the study of the existing robot and communication hardware (e.g., the XBee Pro) to determine the impacts of topology changes on the communication and networking role of each robot; (ii) the implementation of a prototype control interface to identify the requirements to make humans part of a robot swarm; (iii) the development of—simulated and real-world—case studies to evaluate the methodology; finally, (iv) experiments at the EAC Evolvable Lunar Analogue facility or in the context of ESA's CAVES activity to establish the ease of use and the efficiency of the platform. Completing all of these steps will be an important achievement for the realization of collaborative human and robotic exploration of the solar system—a priority of ESA.

The project can also contribute to the advancement of the internet-of-things (IoT) and the development of embedded systems and robotics for the service industry. For example, it would greatly benefit humanitarian companies developing interoperable IoT for disaster areas. The

1. `http://www.esa.int/Our_Activities/Space_Engineering_Technology/Networking_Partnering_Initiative`

ability to optimize the number of robots and their communication links, in fact, allows to reduce deployment costs and to mitigate the logistic challenges of these applications.

Nonetheless, the project is, above all, one of great interest for space exploration: it has the potential to make planetary exploration more effective—and safer—for humans. It fits within the framework of SpaceShip EAC [2]. It is a research effort based on terrestrial technology that will be adapted to space exploration and it will develop low-TRL technologies. These will have vast applicability for space missions such as:

1. The exploration of lava tubes on the Moon. Exploiting the lunar analogue facility at EAC and the caves used by ESA for astronaut training, this project can increase the safety and the productivity of future lunar explorers travelling to the Moon Village.
2. The exploration of Mars surface. The proposed software infrastructure is meant for both indoor and outdoor environments. Connectivity-preserving quad-rotors, on Mars, can support long distance information relays to be used by multiple exploration parties.

2. http://www.esa.int/spaceinvideos/Videos/2016/02/SpaceShip_EAC_heading_for_the_Moon

# REFERENCES


[1] ACHICHE, S., SHLECHTINGEN, M., RAISON, M., BARON, L. and SANTOS, I. F. (2015). Adaptive neuro-fuzzy inference system models for force prediction of a mechatronic flexible structure. *Journal of Integrated Design and Process Science*, 19, 77–94.

[2] AGNE, A., HAPPE, M., KELLER, A., LÜBBERS, E., PLATTNER, B., PLATZNER, M. and PLESSL, C. (2014). Reconos: An operating system approach for reconfigurable computing. *IEEE Micro*, 34, 60–71.

[3] ALDERIGHI, M., CASINI, F., D'ANGELO, S., MANCINI, M., CODINACHS, D., PASTORE, S., SORRENTI, G., STERPONE, L., WEIGAND, R. and VIOLANTE, M. (2008). Robustness analysis of soft error accumulation in sram-fpgas using flipper and star/rora. *Radiation and Its Effects on Components and Systems (RADECS), 2008 European Conference on*. 157–161.

[4] ALEXANDRESCU, D., STERPONE, L. and LÓPEZ-ONGIL, C. (2014). Fault injection and fault tolerance methodologies for assessing device robustness and mitigating against ionizing radiation. *2014 19th IEEE European Test Symposium (ETS)*. 1–6.

[5] ALOUANI, I., MEDIOUNI, B. L. and NIAR, S. (2015). A multi-objective approach for software/hardware partitioning in a multi-target tracking system. *2015 International Symposium on Rapid System Prototyping (RSP)*. 119–125.

[6] ANTONELLI, G., ARRICHIELLO, F., CACCAVALE, F. and MARINO, A. (2014). Decentralized time-varying formation control for multi-robot systems. *The International Journal of Robotics Research*, 33, 1029–1043.

[7] APVRILLE, L., TANZI, T. and DUGELAY, J. L. (2014). Autonomous drones for assisting rescue services within the context of natural disasters. *2014 XXXIth URSI General Assembly and Scientific Symposium (URSI GASS)*. 1–4.

[8] AVIZIENIS, A., LAPRIE, J.-C., RANDELL, B. and LANDWEHR, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1, 11–33.

[9] BAJRACHARYA, M., MAIMONE, M. W. and HELMICK, D. (2008). Autonomy for mars rovers: Past, present, and future. *Computer*, 41, 44–50.

[10] BARAL, C., EITER, T., BJÄRELAND, M. and NAKAMURA, M. (2008). Maintenance goals of agents in a dynamic environment: Formulation and policy construction. *Artificial Intelligence*, 172, 1429 – 1469.

[11] BARTOLINI, D. B., CARMINATI, M., CATTANEO, R., PANERATI, J., SIRONI, F. and SCIUTO, D. (2012). Acos: an autonomic management layer enhancing commodity operating systems. *2nd International Workshop on Computing in Heterogeneous, Autonomous 'N' Goal-oriented Environments*.

[12] BARUAH, S. (2004). Task partitioning upon heterogeneous multiprocessor platforms. *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*. 536–543.

[13] BAYINDIR, L. (2016). A review of swarm robotics tasks. *Neurocomputing*, 172, 292–321.

[14] BELTRAME, G., FOSSATI, L. and SCIUTO, D. (2009). ReSP: a nonintrusive Transaction-Level reflective MPSoC simulation platform for design space exploration. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28, 1857–1869.

[15] BELTRAME, G., FOSSATI, L. and SCIUTO, D. (2010). Decision-theoretic design space exploration of multiprocessor platforms. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29, 1083–1095.

[16] BELTRAME, G. and NICOLESCU, G. (2011). A multi-objective decision-theoretic exploration algorithm for platform-based design. *Proc. of Design, Automation and Test in Europe (DATE)*. In press.

[17] BERTRAND, A. and MOONEN, M. (2013). Distributed computation of the fiedler vector with application to topology inference in ad hoc networks. *Signal Processing*, 93, 1106 – 1117.

[18] BEZZO, N., SORRENTINO, F. and FIERRO, R. (2013). Decentralized estimation of topology changes in wireless robotic networks. *2013 American Control Conference*. 5899–5904.

[19] BOLCHINI, C., CARMINATI, M. and MIELE, A. (2013). Self-adaptive fault tolerance in multi-/many-core systems. *Journal of Electronic Testing*, 29, 159–175.

[20] BOLCHINI, C., CARMINATI, M., MIELE, A., DAS, A., KUMAR, A. and VEERAVALLI, B. (2013). Run-time mapping for reliable many-cores based on energy/performance trade-offs. *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium on*. 58–64.

[21] BRAMBILLA, M., FERRANTE, E., BIRATTARI, M. and DORIGO, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7, 1–41.

[22] BRAMBILLA, M., FERRANTE, E., BIRATTARI, M. and DORIGO, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7, 1–41.

[23] BRUST, M. R. and STRIMBU, B. M. (2015). A networked swarm model for uav deployment in the assessment of forest environments. *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. 1–6.

[24] BURNS, A. and DAVIS, R. (2013). Mixed criticality systems - a review. Technical report, University of York.

[25] CASSANO, L., COZZI, D., KORF, S., HAGEMEYER, J., PORRMANN, M. and STERPONE, L. (2013). On-line testing of permanent radiation effects in reconfigurable systems. *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*. 717–720.

[26] CENSOR, Y. (1977). Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization*, 4, 41–59.

[27] CHEN, C., PANERATI, J. and BELTRAME, G. (2016). Effects of online fault detection mechanisms on probabilistic timing analysis. *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 41–46.

[28] CHEN, C., SANTINELLI, L., HUGUES, J. and BELTRAME, G. (2016). Static probabilistic timing analysis in presence of faults. *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*. 1–10.

[29] CHEN, X., CHANG, L.-W., RODRIGUES, C. I., LV, J., WANG, Z. and HWU, W.-M. (2014). Adaptive cache management for energy-efficient gpu computing. *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, Washington, DC, USA, MICRO-47, 343–355.

[30] COELLO, C. A. (1998). A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1, 269–308.

[31] COELLO, C. A. (2000). An updated survey of ga-based multiobjective optimization techniques. *ACM Comput. Surv.*, 32, 109–143.

[32] CRUZ, P., FIERRO, R. and ABDALLAH, C. (2016). Cooperative learning for robust connectivity in multirobot heterogeneous networks. *Control of Complex Systems: Theory and Applications*, 451–474.

[33] CVETKOVIC, D. and ROWLINSON, P. (2004). Spectral graph theory. *Topics in algebraic graph theory*, 88–112.

[34] CZYZŻAK, P. and JASZKIEWICZ, A. (1998). Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7, 34–47.

[35] DAVIS, R. I. and BURNS, A. (2011). A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43, 35:1–35:44.

[36] DEB, K. and GOEL, T. (2001). Controlled elitist non-dominated sorting genetic algorithms for better convergence. E. Zitzler, L. Thiele, K. Deb, C. Coello Coello and D. Corne, editors, *Evolutionary Multi-Criterion Optimization*, Springer Berlin Heidelberg, vol. 1993 of *Lecture Notes in Computer Science*. 67–81.

[37] DI LORENZO, P. and BARBAROSSA, S. (2014). Distributed estimation and control of algebraic connectivity over random graphs. *IEEE Transactions on Signal Processing*, 62, 5615–5628.

[38] DIEBOLD, F. X., PIAZZESI, M. and RUDEBUSCH, G. (2005). Modeling bond yields in finance and macroeconomics. Working Paper 11089, National Bureau of Economic Research.

[39] DOUGLAS, R. (2014). Insuring Resilience - Employing Approaches from the Re/insurance Sector to Encourage Sustainable Design and Operations against Natural Hazards. *AGU Fall Meeting Abstracts*.

[40] DUMITRIU, V., KIRISCHIAN, L. and KIRISCHIAN, V. (2014). Decentralized runtime recovery mechanism for transient and permanent hardware faults for spaceborne fpga-based computing systems. *Adaptive Hardware and Systems (AHS), 2014 NASA/ESA Conference on*. 47–54.

[41] DURAND, D. and HOBERMAN, R. (2011). Computational genomics and molecular biology, hmm lecture notes. Technical report, Carnegie Mellon University.

[42] ELSAYED, E. A. (2012). *Reliability Engineering*. Wiley Publishing, second edition.

[43] ERBAS, C. (2006). *System-level Modelling and Design Space Exploration for Multiprocessor Embedded System-on-chip Architectures*. Amsterdam University Press.

[44] FEHR, E. (2002). Behavioural science: The economics of impatience. *Nature*, 415, 269–272.

[45] FENDER, I., GIBSON, M. S. and MOSSER, P. C. (2001). An international survey to stress tests. *Current Issues in Economics and Finance*, 10, 1–6.

[46] FODÉ, C., PANERATI, J., DESROCHES, P., VALDATTA, M. and BELTRAME, G. (2015). Monitoring glaciers from space using a cubesat. *IEEE Communications Magazine*, 53, 208–210.

[47] FOLKE, C. (2006). Resilience: The emergence of a perspective for social–ecological systems analyses. *Global Environmental Change*, 16, 253 – 267. Resilience, Vulnerability, and Adaptation: A Cross-Cutting Theme of the International Human Dimensions Programme on Global Environmental ChangeResilience, Vulnerability, and Adaptation: A Cross-Cutting Theme of the International Human Dimensions Programme on Global Environmental Change.

[48] FONSECA, C. M. and FLEMING, P. J. (1995). An overview of evolutionary algorithms in multiobjective optimization. *Evol. Comput.*, 3, 1–16.

[49] FORNACIARI, W., SCIUTO, D., SILVANO, C. and ZACCARIA, V. (2002). A Sensitivity-Based design space exploration methodology for embedded systems. *Design Automation for Embedded Systems*, 7, 7–33.

[50] FROYLAND, E. and LARSEN, K. (2002). How vulnerable are financial institutions to macroeconomic changes? an analysis based on stress testing. *Norges Bank.Economic Bulletin*, 3, 92–98.

[51] GAINES, D., ANDERSON, R., DORAN, G., HUFFMAN, W., JUSTICE, H., MACKEY, R., RABIDEAU, G., VASAVADA, A., VERMA, V., ESTLIN, T., FESQ, L., INGHAM, M., MAIMONE, M. and NESNAS, I. (2016). Productivity challenges for mars rover operations. *The 26th International Conference on Automated Planning and Scheduling*.

[52] GAITANOS, G. C., WILLIAMS, C., BOOBIS, L. H. and BROOKS, S. (1993). Human muscle metabolism during intermittent maximal exercise. *Journal of Applied Physiology*, 75, 712–719.

[53] GAREY, M. R., GRAHAM, R. L. and JOHNSON, D. S. (1977). The complexity of computing steiner minimal trees. *SIAM journal on applied mathematics*, 32, 835–859.

[54] GARVIE, M. and THOMPSON, A. (2004). Scrubbing away transients and jiggling around the permanent: long survival of fpga systems through evolutionary self-repair. *On-Line Testing Symposium, 2004. IOLTS 2004. Proceedings. 10th IEEE International*. 155–160.

[55] GHEDINI, C., RIBEIRO, C. H. C. and SABATTINI, L. (2016). A decentralized control strategy for resilient connectivity maintenance in multi-robot systems subject to failures. *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*. London, UK.

[56] GIVARGIS, T., VAHID, F. and HENKEL, J. (2001). System-level exploration for pareto-optimal configurations in parameterized systems-on-a-chip. *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on*. 25–30.

[57] GOMBOLAY, M., WILCOX, R. and SHAH, J. (2013). Fast Scheduling of Multi-Robot Teams with Temporospatial Constraints. *Robotics: Science and Systems*.

[58] GROTZINGER, J. P., CRISP, J., VASAVADA, A. R., ANDERSON, R. C., BAKER, C. J., BARRY, R., BLAKE, D. F., CONRAD, P., EDGETT, K. S., FERDOWSKI, B., GELLERT, R., GILBERT, J. B., GOLOMBEK, M., GÓMEZ-ELVIRA, J., HASSLER, D. M., JANDURA, L., LITVAK, M., MAHAFFY, P., MAKI, J., MEYER, M., MALIN, M. C., MITROFANOV, I., SIMMONDS, J. J., VANIMAN, D., WELCH, R. V. and WIENS, R. C. (2012). Mars science laboratory mission and science investigation. *Space Science Reviews*, 170, 5–56.

[59] GRUHN, H. and PERSSON, P. (2014). Towards a robust algorithm for distributed monitoring of network topology changes. *2014 13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET)*. 1–7.

[60] GUPTA, S., ANSARI, A., FENG, S. and MAHLKE, S. (2009). Adaptive online testing for efficient hard fault detection. *2009 IEEE Int. Conf. Comput. Des.* IEEE, 343–349.

[61] HAY, J., GUTHRIE, P., MULLINS, C., GRESHAM, E. and CHRISTENSEN, C. (2009). *Global Space Industry: Refining the Definition of "New Space"*, American Institute of Aeronautics and Astronautics.

[62] HEGEDÜS, Á., HORVÁTH, Á. and VARRÓ, D. (2015). A model-driven framework for guided design space exploration. *Automated Software Engineering*, 22, 399–436.

[63] HEIRTZLER, J. (2002). The future of the south atlantic anomaly and implications for radiation damage in space. *Journal of Atmospheric and Solar-Terrestrial Physics*, 64, 1701 – 1708. Space Weather Effects on Technological Systems.

[64] HENKEL, J., EBI, T., AMROUCH, H. and KHDR, H. (2013). Thermal management for dependable on-chip systems. *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*. 113–118.

[65] HEYNDERICKX, D., QUAGHEBEUR, B., SPEELMAN, E. and DALY, E. (2000). Esa's space environment information system (spenvis): a www interface to models of the space environment and its effects. *Proc. AIAA*, 371.

[66] HILL, M. and MARTY, M. (2008). Amdahl's law in the multicore era. *Computer*, 41, 33–38.

[67] HOLLING, C. S. (1973). Resilience and stability of ecological systems. *Ann. Rev. of Ecology and Systematics*, 4, 1–23.

[68] HOLLING, C. S. (2001). Understanding the complexity of economic, ecological, and social systems. *Ecosystems*, 4, 390–405.

[69] HSIEH, M. A., HALÁSZ, Á., BERMAN, S. and KUMAR, V. (2008). Biologically inspired redistribution of a swarm of robots among multiple sites. *Swarm Intelligence*, 2, 121–141.

[70] IBRAHIM, M.-H., PAL, C. and PESANT, G. (2017). Improving probabilistic inference in graphical models with determinism and cycles. *Machine Learning*, 106, 1–54.

[71] INTEL (2004). White paper: Enhanced intel speedstep technology for the intel pentium m processor. `ftp://download.intel.com/design/network/papers/30117401.pdf`.

[72] ISHIBUCHI, H. and MURATA, T. (1996). Multi-objective genetic local search algorithm. *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*. 119–124.

[73] ISHIBUCHI, H. and MURATA, T. (1998). A multi-objective genetic local search algorithm and its application to flowshop scheduling. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 28, 392 –403.

[74] JACKSON, J., FAIED, M., KABAMBA, P. and GIRARD, A. (2013). Distributed constrained minimum-time schedules in networks of arbitrary topology. *IEEE Transactions on Robotics*, 29, 554–563.

[75] JACOBS, A., CIESLEWSKI, G., GEORGE, A. D., GORDON-ROSS, A. and LAM, H. (2012). Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive fpga-based space computing. *ACM Trans. Reconfigurable Technol. Syst.*, 5, 21:1–21:30.

[76] JADDOE, S. and PIMENTEL, A. D. (2008). Signature-Based calibration of analytical System-Level performance models. *Proceedings of the 8th international workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation*. Springer-Verlag, Berlin, Heidelberg, SAMOS '08, 268–278.

[77] JASZKIEWICZ, A. (2004). A comparative study of multiple-objective metaheuristics on the bi-objective set covering problem and the pareto memetic algorithm. *Annals of Operations Research*, 131, 135–158.

[78] JASZKIEWICZ, A. and DĄBROWSKI, G. (2005). MOMH: Multiple Objective Meta Heuristics. available at the web site http://home.gna.org/momh/.

[79] KAELBLING, L. P., LITTMAN, M. L. and MOORE, A. P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 237–285.

[80] KARNIK, T. and HAZUCHA, P. (2004). Characterization of soft errors caused by single event upsets in cmos processes. *Dependable and Secure Computing, IEEE Transactions on*, 1, 128–143.

[81] KARP, R. M. (1972). *Reducibility among Combinatorial Problems*, Springer US, Boston, MA. 85–103.

[82] KATZ, D. S. and SOME, R. R. (2003). Nasa advances robotic space exploration. *Computer*, 36, 52–61.

[83] KENNEDY, J. and EBERHART, R. (1995). Particle swarm optimization. *Neural Networks, 1995. Proceedings., IEEE International Conference on*. vol. 4, 1942–1948 vol.4.

[84] KEPHART, J. O. and CHESS, D. M. (2003). The vision of autonomic computing. *Computer*, 36, 41–50.

[85] KOLLAT, J. B. and REED, P. M. (2005). *The Value of Online Adaptive Search: A Performance Comparison of NSGAII, $\epsilon$-NSGAII and $\epsilon$MOEA*, Springer Berlin Heidelberg, Berlin, Heidelberg. 386–398.

[86] KOLLER, D. and FRIEDMAN, N. (2009). *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press.

[87] KORSAH, G., KANNAN, B., BROWNING, B., STENTZ, A. and DIAS, M. (2012). xBots: An approach to generating and executing optimal multi-robot plans with cross-schedule dependencies. *Proceedings - IEEE International Conference on Robotics and Automation*, 115–122.

[88] KRUGMAN, P. (2009). How did economists get it so wrong? *New York Times*, 2.

[89] KRUPKE, D., ERNESTUS, M., HEMMER, M. and FEKETE, S. P. (2015). Distributed cohesive control for robot swarms: Maintaining good connectivity in the presence of exterior forces. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 413–420.

[90] KWOK, Y.-K. and AHMAD, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31, 406–471.

[91] L. LUO, CHAKRABORTY, N. and SYCARA, K. (2011). Multi-robot assignment algorithm for tasks with set precedence constraints. *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2526–2533.

[92] LIU, C. L. and LAYLAND, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20, 46–61.

[93] LO IUDICE, F., GAROFALO, F. and SORRENTINO, F. (2015). Structural permeability of complex networks to control signals. *Nature Communications*, 6, 8349 EP –.

[94] LUKASIEWYCZ, M., GLAY, M., HAUBELT, C. and TEICH, J. (2008). Efficient symbolic multi-objective design space exploration. *ASP-DAC '08: Proceedings of the 2008 Asia and South Pacific Design Automation Conference*. IEEE Computer Society Press, Seoul, Korea, 691–696.

[95] MAGGIO, M., HOFFMANN, H., PAPADOPOULOS, A. V., PANERATI, J., SANTAMBROGIO, M. D., AGARWAL, A. and LEVA, A. (2012). Comparison of decision-making strategies for self-optimization in autonomic computing systems. *ACM Trans. Auton. Adapt. Syst.*, 7, 36:1–36:32.

[96] MARIANI, G., BRANKOVIC, A., PALERMO, G., JOVIC, J., ZACCARIA, V. and SILVANO, C. (2010). A correlation-based design space exploration methodology for multi-processor systems-on-chip. *Design Automation Conference (DAC), 2010 47th ACM/IEEE*. 120–125.

[97] MARKOWITZ, H. (1952). Portfolio selection*. *The Journal of Finance*, 7, 77–91.

[98] MARLER, R. and ARORA, J. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26, 369–395.

[99] MARTIN, G. (2006). Overview of the MPSoC design challenge. *Design Automation Conference, 2006 43rd ACM/IEEE*. 274–279.

[100] MAURER, R. H., FRAEMAN, M. E., MARTIN, M. N. and ROTH, D. R. (2008). Harsh environments: Space radiation environment, effects, and mitigation. *Johns Hopkins APL Technical Digest*, 28, 17–29.

[101] MEDIOUNI, B. L., NIAR, S., BENMANSOUR, R., BENATCHBA, K. and KOUDIL, M. (2015). A bi-objective heuristic for heterogeneous mpsoc design space exploration. *2015 10th International Design Test Symposium (IDT)*. 90–95.

[102] METE, H. O. and ZABINSKY, Z. B. (2010). Stochastic optimization of medical supply location and distribution in disaster management. *International Journal of Production Economics*, 126, 76 – 84. Improving Disaster Supply Chain Management – Key supply chain factors for humanitarian relief.

[103] MEYER, B., HARTMAN, A. and THOMAS, D. (2010). Cost-effective slack allocation for lifetime improvement in noc-based mpsocs. *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*. 1596–1601.

[104] MIETTINEN, K. and MÄKELÄ, M. M. (2002). On scalarizing functions in multiobjective optimization. *OR Spectrum*, 24, 193–213.

[105] MOHANTY, S., PRASANNA, V. K., NEEMA, S. and DAVIS, J. (2002). Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. *SIGPLAN Not.*, 37, 18–27.

[106] NIEMEIER, M., WIESE, A. and BARUAH, S. (2011). Partitioned real-time scheduling on heterogeneous shared-memory multiprocessors. *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*. 115–124.

[107] NUNES, E. and GINI, M. (2015). Multi-Robot Auctions for Allocation of Tasks with Temporal Constraints. *AAAI*, 2110–2116.

[108] NUNES, E., MANNER, M., MITICHE, H. and M.GINI (2016). A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, 1–45.

[109] OKABE, T., JIN, Y. and SENDHOFF, B. (2003). A critical survey of performance indices for multi-objective optimisation. *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*. vol. 2, 878–885 Vol.2.

[110] ÖZVEREN, C. M., WILLSKY, A. S. and ANTSAKLIS, P. J. (1991). Stability and stabilizability of discrete event dynamic systems. *J. ACM*, 38, 729–751.

[111] PALERMO, G., SILVANO, C. and ZACCARIA, V. (2008). Discrete particle swarm optimization for multi-objective design space exploration. *Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on*. 641–644.

[112] PALERMO, G., SILVANO, C. and ZACCARIA, V. (2009). ReSPIR: a response Surface-Based pareto iterative refinement for Application-Specific design space exploration. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28, 1816–1829.

[113] PALESI, M. and GIVARGIS, T. (2002). Multi-objective design space exploration using genetic algorithms. *CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign*. ACM, Estes Park, Colorado, 67–72.

[114] PANERATI, J. (2012). *Enhancing Self-Adaptive Computing Systems via Artificial Intelligence Techniques and Active Learning*. Master's thesis, University of Illinois at Chicago.

[115] PANERATI, J., ABDI, S. and BELTRAME, G. (2014). Balancing system availability and lifetime with dynamic hidden markov models. *Adaptive Hardware and Systems (AHS), 2014 NASA/ESA Conference on*. 240–247.

[116] PANERATI, J. and BELTRAME, G. (2014). A comparative evaluation of multi-objective exploration algorithms for high-level design. *ACM Trans. Des. Autom. Electron. Syst.*, 19, 15:1–15:22.

[117] PANERATI, J. and BELTRAME, G. (2015). Trading off power and fault-tolerance in real-time embedded systems. *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 1–8.

[118] PANERATI, J., BELTRAME, G., SCHWIND, N., ZELTNER, S. and INOUE, K. (2016). Probabilistic resilience in hidden markov models. *IOP Conference Series: Materials Science and Engineering*, 131, 012007.

[119] PANERATI, J., GIANOLI, L. G., PINCIROLI, C., SHABAH, A., NICOLESCU, G. and BELTRAME, G. (2017). From swarms to stars – task coverage in robot swarms with connectivity constraints [under review]. *Autonomous Robots*.

[120] PANERATI, J., MAGGIO, M., CARMINATI, M., SIRONI, F., TRIVERIO, M. and SANTAMBROGIO, M. D. (2014). Coordination of independent loops in self-adaptive systems. *ACM Trans. Reconfigurable Technol. Syst.*, 7, 12:1–12:16.

[121] PANERATI, J., SCHWIND, N., ZELTNER, S., INOUE, K. and BELTRAME, G. (2017). Assessing the resilience of stochastic dynamic systems under partial observability [under review]. *Science Advances*.

[122] PANERATI, J., SIRONI, F., CARMINATI, M., MAGGIO, M., BELTRAME, G., GMYTRASIEWICZ, P. J., SCIUTO, D. and SANTAMBROGIO, M. D. (2013). On self-adaptive resource allocation through reinforcement learning. *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013)*. 23–30.

[123] PAULIN, P. and KNIGHT, J. (1989). Algorithms for high-level synthesis. *Design & Test of Computers, IEEE*, 6, 18–31.

[124] PEARCE, L. (2003). Disaster management and community planning, and public participation: How to achieve sustainable hazard mitigation. *Natural Hazards*, 28, 211–228.

[125] PECORA, L. M. and CARROLL, T. L. (1998). Master stability functions for synchronized coupled systems. *Phys. Rev. Lett.*, 80, 2109–2112.

[126] PECORA, L. M., SORRENTINO, F., HAGERSTROM, A. M., MURPHY, T. E. and ROY, R. (2014). Cluster synchronization and isolated desynchronization in complex networks with symmetries. *Nature Communications*, 5, 4079 EP –.

[127] PERISTIANI, S., MORGAN, D. P. and SAVINO, V. (2010). The information value of the stress test and bank opacity. *FRB of New York Staff Report*.

[128] PETERSEN, E. (2011). *Single event effects in aerospace*. Wiley-IEEE Press.

[129] PETRICK, D., ESPINOSA, D., RIPLEY, R., CRUM, G., GEIST, A. and FLATLEY, T. (2014). Adapting the reconfigurable spacecube processing system for multiple mission applications. *2014 IEEE Aerospace Conference*. 1–20.

[130] PIMENTEL, A., ERBAS, C. and POLSTRA, S. (2006). A systematic approach to exploring embedded system architectures at multiple abstraction levels. *Computers, IEEE Transactions on*, 55, 99–112.

[131] PINCIROLI, C. and BELTRAME, G. (2016). Buzz: An extensible programming language for heterogeneous swarm robotics. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016)*. IEEE Computer Society Press, Los Alamitos, CA. In press.

[132] PINCIROLI, C., LEE-BROWN, A. and BELTRAME, G. (2016). A tuple space for data sharing in robot swarms. *proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 287–294.

[133] PINCIROLI, C., TRIANNI, V., O'GRADY, R., PINI, G., BRUTSCHY, A., BRAMBILLA, M., MATHEWS, N., FERRANTE, E., DI CARO, G., DUCATELLE, F., BIRATTARI, M., GAMBARDELLA, L. M. and DORIGO, M. (2012). Argos: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6, 271–295.

[134] PONDA, S., JOHNSON, L., KOPEIKIN, A., CHOI, H. and HOW, J. (2012). Distributed planning strategies to ensure network connectivity for dynamic heterogeneous teams. *IEEE Journal on Selected Areas in Communications*, 30, 861–869.

[135] PONDA, S., REDDING, J., HAN-LIM, C., HOW, J., VAVRINA, M. and VIAN, J. (2010). Decentralized planning for complex missions with dynamic communication constraints. *Proceedings of the 2010 American Control Conference*. IEEE, 3998–4003.

[136] QIN, X. and JIANG, H. (2005). A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. *Journal of Parallel and Distributed Computing*, 65, 885 – 900.

[137] RABINER, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77, 257–286.

[138] RARAVI, G., ANDERSSON, B., BLETSAS, K. and NELIS, V. (2012). Outstanding paper award: Task assignment algorithms for two-type heterogeneous multiprocessors. *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*. 34–43.

[139] REYNOLDS, C. W. (1999). Steering behaviors for autonomous characters.

[140] ROSS, S. M. (2006). *Introduction to Probability Models, Ninth Edition*. Academic Press, Inc., Orlando, FL, USA.

[141] ROTA, G.-C. (1964). The number of partitions of a set. *American Mathematical Monthly*, 498–504.

[142] RUBENSTEIN, M., CORNEJO, A. and NAGPAL, R. (2014). Programmable self-assembly in a thousand-robot swarm. *Science*, 345, 795–799.

[143] RUSSELL, S. and NORVIG, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition.

[144] RUSSELL, S. J. and NORVIG, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, second edition.

[145] SAHAI, T., SPERANZON, A. and BANASZUK, A. (2012). Hearing the clusters of a graph: A distributed algorithm. *Automatica*, 48, 15–24.

[146] SALDAÑA, D., PROROK, A., CAMPOS, M. F. and KUMAR, V. (2016). Triangular networks for resilient formations. *13th International Symposium on Distributed Autonomous Robotic Systems (DARS)*.

[147] SCHUCK, C., HAETZER, B. and BECKER, J. (2011). Reconfiguration techniques for self-x power and performance management on xilinx virtex-ii/virtex-ii-pro fpgas. *Int. J. Reconfig. Comput.*, 2011, 1:1–1:12.

[148] SCHWIND, N., MAGNIN, M., INOUE, K., OKIMOTO, T., SATO, T., MINAMI, K. and MARUYAMA, H. (2016). Formalization of resilience for constraint-based dynamic systems. *Journal of Reliable Intelligent Environments*, 2, 17–35.

[149] SCHWIND, N., OKIMOTO, T., INOUE, K., CHAN, H., RIBEIRO, T., MINAMI, K. and MARUYAMA, H. (2013). Systems resilience: A challenge problem for dynamic constraint-based agent systems. *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*. AAMAS '13, 785–788.

[150] SERAFINI, P. (1994). Simulated annealing for multi objective optimization problems. G. Tzeng, H. Wang, U. Wen and P. Yu, editors, *Multiple Criteria Decision Making*, Springer New York. 283–292.

[151] SHAO, Y. S., REAGEN, B., WEI, G. Y. and BROOKS, D. (2014). Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 97–108.

[152] SHELDON, D., VAHID, F. and LONARDI, S. (2007). Soft-core processor customization using the design of experiments paradigm. *DATE Conference, 2007*. 1–6.

[153] SIDES, C. H. (1999). *How to write and present technical information*. Cambridge University Press.

[154] SIMS, C. A. (1980). Macroeconomics and reality. *Econometrica*, 48, 1–48.

[155] SIVANANDAM, S. N. and DEEPA, S. N. (2007). *Introduction to Genetic Algorithms*. Springer Publishing Company, Incorporated, first edition.

[156] SMOLENS, J. C., GOLD, B. T., HOE, J. C., FALSAFI, B. and MAI, K. (2007). Detecting emerging wearout faults. *In Proceedings of the IEEE Workshop on Silicon Errors in Logic - System Effects*.

[157] SOLEYMANI, T., GARONE, E. and DORIGO, M. (2015). Distributed constrained connectivity control for proximity networks based on a receding horizon scheme. *2015 American Control Conference (ACC)*. 1369–1374.

[158] SOLTERO, D. E., SCHWAGER, M. and RUS, D. (2014). Decentralized path planning for coverage tasks using gradient descent adaptive control. *The International Journal of Robotics Research*, 33, 401–425.

[159] SRINIVAS, N. and DEB, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evol. Comput.*, 2, 221–248.

[160] STØY, K. (2001). Using situated communication in distributed autonomous mobile robotics. *Proceedings of the Seventh Scandinavian Conference on Artificial Intelligence*. IOS Press, Amsterdam, The Netherlands, The Netherlands, SCAI '01, 44–52.

[161] TAGHAVI, T. and PIMENTEL, A. D. (2011). Design metrics and visualization techniques for analyzing the performance of moeas in dse. *ICSAMOS*. 67–76.

[162] TAN, Y., LIU, W. and QIU, Q. (2009). Adaptive Power Management Using Reinforcement Learning. *Proceedings of the 2009 International Conference on Computer-Aided Design*. 461–467.

[163] TRIANNI, V. and DORIGO, M. (2006). Self-organisation and communication in groups of simulated and physical robots. *Biological Cybernetics*, 95, 213–231.

[164] TYLKA, A. J., ADAMS, J. H., BOBERG, P. R., BROWNSTEIN, B., DIETRICH, W. F., FLUECKIGER, E. O., PETERSEN, E. L., SHEA, M. A., SMART, D. F. and SMITH, E. C. (1997). Creme96: A revision of the cosmic ray effects on micro-electronics code. *IEEE Transactions on Nuclear Science*, 44, 2150–2160.

[165] ULUNGU, E., TEGHEM, J., FORTEMPS, P. and TUYTTENS, D. (1999). Mosa method: a tool for solving multiobjective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis*, 8, 221–236.

[166] ULUNGU, E. L. and TEGHEM, J. (1994). Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3, 83–104.

[167] VEDANT, PANERATI, J. and BELTRAME, G. (2014). An orbit-specific fault-injector to assess fault-mitigation strategies in space fpgas. *SEFUW: SpacE FPGA Users Workshop, 2nd Edition*. ESTEC.

[168] WACHS, M. and WHITE, D. (1991). p, q-stirling numbers and set partition statistics. *Journal of Combinatorial Theory, Series A*, 56, 27 – 46.

[169] WALKER, B., HOLLING, C. S., CARPENTER, S. R. and KINZIG, A. (2004). Resilience, adaptability and transformability in social-ecological systems. *Ecology and Society*, 9.

[170] WAYDO, S., HENRY, D. and CAMPBELL, M. (2002). Cubesat design for leo-based earth science missions. *Proceedings, IEEE Aerospace Conference*. vol. 1, 1–435–1–445 vol.1.

[171] WESTE, N. and HARRIS, D. (2010). *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, fourth edition.

[172] XIAO, L., BOYD, S. and KIM, S.-J. (2007). Distributed average consensus with least-mean-square deviation. *Journal of Parallel and Distributed Computing*, 67, 33 – 46.

[173] YAZICI, A., KIRLIK, G., PARLAKTUNA, O. and SIPAHIOGLU, A. (2014). A dynamic path planning approach for multirobot sensor-based coverage considering energy constraints. *IEEE Transactions on Cybernetics*, 44, 305–314.

[174] ZACCARIA, V., PALERMO, G., CASTRO, F., SILVANO, C. and MARIANI, G. (2010). Multicube explorer: An open source framework for design space exploration of chip multi-processors. *2PARMA: Proceedings of the Workshop on Parallel Programming and Run-time Management Techniques for Many-core Architectures*. Hannover, Germany.

[175] ZAREH, M., SABATTINI, L. and SECCHI, C. (2016). Distributed laplacian eigenvalue and eigenvector estimation in multi-robot systems. *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*. London, UK.

[176] ZAREH, M., SABATTINI, L. and SECCHI, C. (2016). Enforcing biconnectivity in multi-robot systems. *CoRR*, abs/1608.02286.

[177] ZEESHAN, M., ALI, A., NAVEED, A., LIU, A. X., WANG, A. and QURESHI, H. K. (2016). Modeling packet loss probability and busy time in multi-hop wireless networks. *EURASIP Journal on Wireless Communications and Networking*, 2016, 168.

[178] ZELAZO, D., FRANCHI, A., BÜLTHOFF, H. H. and GIORDANO, P. R. (2015). Decentralized rigidity maintenance control with range measurements for multi-robot systems. *The International Journal of Robotics Research*, 34, 105–128.

[179] ZHANG, H. and SUNDARAM, S. (2012). Robustness of complex networks with implications for consensus and contagion. *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. 3426–3432.

[180] ZITZLER, E., DEB, K. and THIELE, L. (1999). Comparison of Multiobjective Evolutionary Algorithms: Empirical Results (Revised Version). TIK Report 70, Computer Engineering and Networks Laboratory (TIK), ETH Zurich.

[181] ZITZLER, E., DEB, K. and THIELE, L. (1999). Comparison of Multiobjective Evolutionary Algorithms on Test Functions of Different Difficulty. *Genetic and Evolutionary Computation Conference (GECCO 1999): Bird-of-a-feather Workshop on Multicriterion Optimization*.

[182] ZITZLER, E. and THIELE, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *Evolutionary Computation, IEEE Transactions on*, 3, 257 –271.

# APPENDIX A *POLYORBITE* AND THE CANADIAN SATELLITE DESIGN CHALLENGE

This appendix outlines the research, engineering, and educational, efforts made in the context of Polytechnique Montréal's technical society (*société technique*) *PolyOrbite*. Two of the most relevant papers I co-authored within *PolyOrbite* are briefly recapitulated. At the end of the appendix, the reader can find a short summary of the history of the society.

The paper summarized below was written during my initial year in the MIST Laboratory. As the only graduate student contributing to the study, my responsibilities included: (i) writing the abstract that was originally accepted for presentation; (ii) reviewing and writing about the state of the art; and (iii) reviewing and editing the final document.

**Title:** 3U CubeSat for the Canadian Satellite Design Challenge: A Polytechnique Montréal and University Of Bologna Cooperation

**Summary:** *PolyOrbite* was founded at the end of 2012 and established collaborations with *Université de Montréal* and University of Bologna in early 2013. This conference paper, presented by Mark Smyth at the 64th IAC in September 2013 was the first bit of research released by the technical society. The document reports on the scientific relevance of two payloads developed for a 3U CubeSat and it introduces the technical details of their implementations. The first payload, created by the team located in Bologna, is an autonomous de-orbiting system consisting of a drag-sail. The sail is realized using a shape-memory polymer and the payload, as a whole, is conceived as a plug-and-play device. This technology has the potential to simplify the quick de-orbiting of small satellites, thus, contributing to mitigate the problem represented by low Earth orbit debris. The second payload is an imaging system meant to monitor the Canadian Arctic and investigate changes in its biodiversity. The organizational, educational, and outreach aspects of the project are also presented.

**Presented At:** The 64th International Astronautical Congress (IAC), 23-27 September 2013, Beijing, China.

**URL:** `https://iafastro.directory/iac/archive/browse/IAC-13/E2/3-V.4/18640/`

**Authors:** Mark Smyth, Étienne Bourbeau, Jacopo Panerati, Niccolò Bellini, Alexandra Labbé, Anthony Buffet, Alexandre Guay, Alfredo Locarini, Stefano Naldi, Davide Rastelli, and Marcello Valdatta

The second contribution published by *PolyOrbite* and worth mentioning here is a short paper regarding the team's experience in the Canadian Satellite Design Challenge (CSDC) over the 2012–2014 period. The article appeared in the IEEE Communication Magazine in 2015.

Once again, I was the sole graduate student contributing to the article and I took responsibility for: (i) structuring the document; (ii) summarizing content from multiple sources, previous publications, and other written material; (iii) reviewing and editing the overall document; and (iv) interacting with the editorial staff.

**Title:** Monitoring Glaciers from Space Using a Cubesat

**Summary:** One of the most striking aspects of *PolyOrbite* is its success in the advancement of technically challenging projects despite its spontaneously shaped—and constantly liquid—student-led organization. During the 2012–2014 CSDC campaign, in particular, many practical adversities had to be overcome, e.g., the limited size and experience of the team and the uncertainties associated with the intercontinental collaboration with University of Bologna. Eventually, *PolyOrbite* reached the vibration testing stage of the CSDC (at the Canadian Space Agency's David Florida Laboratory, in Ottawa). When the results were revealed, *PolyOrbite* obtained the lowest step on the podium of the competition—a remarkable achievement for a first attempt. The accomplishment led us to this invited contribution in the IEEE Communication Magazine. The article, intended for a lay audience, reviews the main features of *PolyOrbite* and its history: the foundation at the hand of a group of students from Polytechnique Montréal willing to participate in the second edition of the Canadian Satellite Design Challenge, the contribution of University of Bologna, the satellite's payloads, and the educational impact.

**Published In:** IEEE Communications Magazine, vol. 53, no. 5, pp. 208-210, May 2015.

**DOI:** `https://doi.org/10.1109/MCOM.2015.7105665`

**Authors:** Constance Fodé, Jacopo Panerati, Prescilia Desroches, Marcello Valdatta, and Giovanni Beltrame

## *PolyOrbite*'s CSDC Participations and Designs

To conclude the appendix, *PolyOrbite*'s history is recapitulated in this section. *PolyOrbite* is a technical society composed—and led—by students of Polytechnique Montréal. In late 2012, the original team was born from the spontaneous association of a handful of students willing to participate in the 2012–2014 Canadian Satellite Design Challenge. The CSDC is an inter-university competition for student teams having as objective the development of a 3U CubeSat. From the time of its foundation, until the end of 2016, I was the leader of the team developing the on-board computers of *PolyOrbite*'s satellites. The following subsections are dedicated to the three two-year periods corresponding to the second, third, and fourth iteration of the Canadian Satellite Design Challenge, respectively.

### 2012–2014 Canadian Satellite Design Challenge: Eleonora and Collaboration with University of Bologna

*PolyOrbite*'s experience in the CSDC started with the second iteration of the competition. The first CSDC launched in January 2011 to conclude in 2012 with the victory of the satellite design proposed by team Space Concordia from Concordia University. After this success, one of the students that had led the endeavour, Nick Sweet, presented the CSDC and his experience in a talk at Polytechnique Montréal. Inspired by the episode, a small group of undergraduate students—and the author of this thesis—decided to organize a team to represent Polytechnique Montréal in the second, upcoming, CSDC.

2013 was stocked with many significant events for *PolyOrbite*. First, the team officially became one of the 17 *sociétés techniques* of Polytechnique Montréal. Then, it established short- and long-range collaborations with *Université de Montréal*'s Geocryolab and University of Bologna, respectively. These partnerships proved to be of crucial importance for the definition and implementation of the two payloads that *PolyOrbite* incorporated in its 3U CubeSat. Geocryolab provided the scientific rationale for the imaging system of the satellite as an instrument to monitor changes in the Canadian Arctic. University of Bologna designed a compact de-orbiting system for which *PolyOrbite* received plenty of praise.

The design of the command and data handling system included two computing boards: one acquired from Pumpkin Inc. and hosting a PIC24 micro-controller; and one, internally developed, supporting three different FPGA fabrics. After the final hurdle of the CSDC, i.e., vibration testing, *PolyOrbite*'s satellite Eleonora obtained the third place overall, behind University of Victoria and Concordia University.

### 2014–2016 Canadian Satellite Design Challenge: Hathor and 2016 Intercollegiate Rocket Engineering Competition

The third iteration of the CSDC—*PolyOrbite*'s second participation—was marked by the end of the collaboration with University of Bologna and the forfeiture of its drag-sail. The introduction of a mandatory de-orbiting system as a requirement of the CSDC led the team to select a four-thruster ion propeller as the new primary payload (IonDrop). A second payload, consisting of an autonomous greenhouse (SpaceBean), was also included in the design of *PolyOrbite*'s second 3U CubeSat, Hathor.

For the command and data handling system, we decided to re-use Eleonora's Pumpkin Inc. PIC24 design, paired with distributed computing resources in selected sub-systems (e.g. ADCS). Like its predecessor, Hathor went through vibration testing successfully and finished third overall. *PolyOrbite* also won CSDC's Educational Outreach prize.

Additionally, in 2016, *PolyOrbite* collaborated with rocket-building technical society Oronos. The common objective was the development of a scientific mission for the SDL Payload Challenge of the 2016 Intercollegiate Rocket Engineering Competition in Utah. The selected experiment was the study of the deformation of Eleonora's 3U structure during launch on a sounding rocket. The payload integrated a Raspberry Pi 2 B computer with SenseHat, an analog-to-digital converter, and a CAN bus interface. Unfortunately, the rocket experienced rapid unscheduled disassembly on the day of the competition.

### 2016–2018 Canadian Satellite Design Challenge: ORU-S and Collaboration with the Canadian Space Agency

The ongoing iteration of the CSDC started with *PolyOrbite* choosing to retain its electric propulsion system, IonDrop, as one of two payloads (and de-orbiting device). SpaceBean's greenhouse was initially set aside, and new proposals were evaluated.

However, as the opportunity for a collaboration that involved *PolyOrbite*, the Canadian Space Agency, and Polytechnique Montréal appeared, SpaceBean was reintegrated as a secondary payload and a third objective was added to the project. The third mission of the 3U CubeSat, christened ORU-S, consists in the validation of a self-adaptive on-board computer, loosely inspired by the FPGA-based computing board of the 2012–2014 design.

# APPENDIX B MODELLING OF THE ERRORS INDUCED BY SPACE RADIATION FOR FAULT-INJECTION IN FPGAS

This appendix reports on the development of a comprehensive software framework called MORFIN (Mistlab ORbit-specific Fault INjector). MORFIN is a tool that can model the space radiation environment and inject errors into a field-programmable gate array (FPGA) so that they are consistent with a user-specified mission profile.

This research was conducted in the MIST Laboratory of Polytechnique Montréal in 2014. The main author, designer, and developer of the project is Vedant. My co-author responsibilities included: (i) writing parts of the paper; (ii) preparing the illustration; and (iii) reviewing and editing. The work was presented by professor Giovanni Beltrame at ESTEC, Noordwijk, Netherlands, in September 2014.

**Title:** An Orbit-specific Fault-injector to Assess Fault-mitigation Strategies in FPGA-based Computing Systems for Aerospace

**Summary:** As it was mentioned numerous times, the space environment is especially demanding for electronics due to the presence of particle and ionizing radiation. Reconfigurable FPGAs are the ideal test-bench—and a likely implementation platform—for the adaptive methodologies we proposed in [115, 117]. Therefore, the capability of injecting into an FPGA errors (e.g., bit-flips) that are representative of a specific orbit, mission profile, or portion of space becomes an essential prerequisite for pre-flight validation. Various models of the space radiation environment have been developed in the literature and a few of them are publicly available as web services, e.g., SPENVIS [65] and CREME96 [164]. The engineering challenge, here, resides in automating the process that—from the two-line element set of a satellite—injects errors into an FPGA as if it was on-board the spacecraft. The paper presents the detail of how this is achieved. This framework (one of the firsts of its kind, inspired by the work in [75]) offers a ready-to-use and realistic simulated environment for the validation of adaptive and fault-tolerant technologies for aerospace.

**Presented At:** SEFUW: SpacE FPGA Users Workshop, 2nd Edition, ESTEC, Noordwijk, Netherlands.

**URL:** `https://indico.esa.int/indico/event/59/session/5/contribution/17`

**Authors:** Vedant, Jacopo Panerati, and Giovanni Beltrame

# APPENDIX C ONLINE FAULT DETECTION AND PROBABILISTIC TIMING ANALYSIS

This last appendix acknowledge the work conducted on the static probabilistic timing analysis (SPTA) of computing systems using the fault detection mechanisms introduced in [115]. This research was carried out in the MIST Laboratory of Polytechnique Montréal between the end of 2015 and the beginning of 2017. The main author and architect of the investigation is Chao Chen.

My co-author responsibilities included: (i) an advisory role on probabilistic modelling and the fault-detection techniques; (ii) preparing some of the figures; and (iii) reviewing and editing the paper. An extended version of this work, with title "Probabilistic Timing Analysis of Random Caches with Fault Detection Mechanisms", has been submitted to the ACM Transactions on Emerging Topics in Computing (TETC) and it is currently under review.

**Title:** Static Probabilistic Timing Analysis with a Permanent Fault Detection Mechanism

**Summary:** In real-time systems, the estimation of the worst case execution time (WCET) of a software task is primordial for the assessment of its schedulability. However, WCET estimates are often overly pessimistic and hinder ideal performance. Random cache memories and probabilistic timing analysis offer a way to circumvent this issue. By randomizing the use of cache blocks and computing the probability distribution of executions times, one can estimate a probabilistic WCET (pWCET), i.e. an execution time that is guaranteed not to be exceeded with a certain probability $p$. In this paper, the problematic is extended with the introduction of transient and permanent faults (often caused by the effects of space radiation [115, 117]). We equip the system under study with the fault detection mechanisms from [115]—to detect and disable those cache blocks that are permanently damaged—and we conduct our SPTA on the overall system to assess its performance. The experimental results support the claim of superior performance provided by dynamic hidden Markov models.

**Published In:** Proceedings of the 2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT).

**DOI:** `https://doi.org/10.1109/DFT.2016.7684067`

**Authors:** Chao Chen, Jacopo Panerati, and Giovanni Beltrame